

Reinforcement Learning Evaluation for Dependent Task Offloading in Mobile Edge Computing Systems

Lubna Thair^{1*}, Awos Kh. Ali²⁾

^{1,2)}University of Mosul, Iraq

lubna.24esp7@student.uomosul.edu.iq, a.k.ali@uomosul.edu.iq

Submitted : Jul 19, 2026 | Accepted : Aug 24, 2026 | Published : Oct 5, 2026

Abstract: Dependent-task directed acyclic graphs require each task to execute locally or offload to a mobile edge computing server, coupling latency, communication cost, and device energy. This study reports a controlled evaluation of deep reinforcement learning for offloading under a common simulator, dataset, and train/validation/final-test protocol. Baselines comprise a recurrent policy with proximal policy optimization (PPO-LSTM) and a double deep Q-network; extensions comprise a Transformer policy, discrete soft actor-critic, and stabilized PPO with advantage normalization, scheduled optimization, divergence-based early stopping, and orthogonal initialization. Agents and five heuristics are evaluated on final-test graphs of ten to fifty tasks under latency-oriented and energy-aware quality of experience (QoE) objectives, with five seeds and Holm-corrected tests. Stabilized PPO attained mean QoE of 0.31–0.43 across task-size and bandwidth analyses, though not at every size or bandwidth mean, exceeding Heterogeneous Earliest Finish Time (HEFT) by 0.05 to 0.15 absolute, yet its gain over PPO-LSTM remained about 0.001 and was not Holm-significant at any bandwidth. Both PPO variants reached the validation threshold at median update five, against ten for the Transformer. Ablation identified no single driver: only removing advantage normalization produced a Holm-significant drop, and only under the energy-aware objective. The Transformer remained competitive, whereas discrete soft actor-critic reduced energy rather than latency. Matching the recurrent baseline in size and inference time, stabilization remains viable on resource-constrained devices but cannot replace retraining on much slower links.

Keywords: Deep reinforcement learning, Directed acyclic graph, Mobile edge computing, Quality of experience, Task offloading

INTRODUCTION

Intelligent mobile applications, including augmented reality, video analytics, navigation, gaming, and autonomous services, generate computation-intensive workloads that exceed the processing and energy capabilities of resource-constrained devices. Mobile edge computing (MEC) addresses this limitation by placing computing resources close to users, so that heavy computation can be offloaded from a mobile device to a nearby edge server (Sivasakthi & Gunasekaran, 2022; Yang et al., 2024).

Many such applications are not monolithic but consist of dependent tasks in a directed acyclic graph (DAG), where each task consumes the outputs of its predecessors. Executing a task locally saturates the device processor and drains the battery, while offloading it to the edge server adds communication delay and congests the wireless link. The optimal offloading strategy is a binary local or remote decision for each task and is dependent on the graph topology, task data size, the available bandwidth, and the relative importance the user assigns to latency versus energy. Computing such plans optimally is NP-hard in general, which has motivated both list-scheduling heuristics such as the Heterogeneous Earliest Finish Time (HEFT) algorithm (Topcuoglu et al., 2002) and, more recently, learning-based approaches.

Among closely related studies, (Wang et al., 2022) used a HEFT-ranked sequence-to-sequence deep reinforcement-learning formulation and showed that learned policies can outperform heuristic scheduling on dependent-task graphs; however, the study considered a single edge-server setting and did not isolate architecture or training-stabilization effects. (Pang et al., 2024) applied Double DQN to dependency-aware vehicle-edge-cloud

*name of corresponding author



This is anCreative Commons License This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

offloading under a joint latency–energy objective and reported DAG-aware gains over heuristic baselines, but remained limited to a value-based agent without a policy-gradient comparison. (Ju et al., 2023) trained a deep reinforcement-learning pointer-network scheduler and optimized delay-constrained throughput, but did not address joint latency–energy quality of experience.

Despite considerable progress, several gaps remain. Earlier methods often rely on fixed heuristics that cannot adapt the latency-energy trade-off, or adopt a single reinforcement learning architecture without isolating the function approximator, the algorithm family, and training stabilization. A controlled comparison of these factors under one pipeline is still missing.

This study evaluates dependency-aware task offloading using reinforcement learning under a unified simulator and an identical quality-of-experience (QoE) scoring path in a single mobile-edge-server environment. It is undertaken as a controlled re-implementation and extension study of the dependent-task offloading framework of Wang et al. (2022) which proposed a sequence-to-sequence long short-term memory (LSTM) policy trained with proximal policy optimization (PPO) (Schulman et al., 2017) and a double deep Q-network (Double DQN) agent (Van Hasselt et al., 2016). The original scheduling and energy simulator is reused without modification so that all methods remain numerically comparable, while every learning component is re-implemented independently.

Accordingly, this study addresses the following research questions. First, how do policy-gradient, value-based, and maximum-entropy reinforcement learning methods compare under an identical dependent-task offloading environment? Second, which stabilization components contribute most to the performance and convergence of proximal policy optimization? Third, how robust are the learned policies to changes in task-graph size and wireless bandwidth? Finally, what computational overhead is introduced by each learning architecture during training and inference? The primary contribution of this study is methodological: it provides a controlled and reproducible evaluation that isolates the effects of the function approximator, the reinforcement-learning family, and training stabilization under identical conditions, rather than proposing a new MEC architecture. Recurrent and attention-based policies are compared with the same trainer, proximal policy optimization is examined by component-wise ablation, and Holm-corrected paired tests are used to report quality of experience, convergence, model size, and inference time.

LITERATURE REVIEW

The theoretical basis is threefold. Mobile edge computing offloads computation from a resource-constrained device to a nearby server over a wireless link. Dependent applications are directed acyclic graphs: tasks obey precedence, and HEFT ranks them by upward rank, which is both the heuristic baseline and the decoding order of every learned policy. Reinforcement learning then maps each ranked task to a binary local-or-offload action, using on-policy PPO (recurrent or Transformer), value-based Double DQN, or off-policy discrete SAC, optionally with PPO stabilization.

Mobile and vehicular edge computing move work from distant clouds to servers near users, creating offloading decisions under limited bandwidth and server capacity (Hu & Huang, 2022; Tang et al., 2023; Uddin et al., 2024; Zhang et al., 2022)

Heuristic schedulers such as HEFT rank tasks by upward cost and assign each task to the processor with the earliest finish time (Topcuoglu et al., 2002). They remain energy-oblivious and cannot adapt a latency-energy trade-off as user preferences change.

A second family treats DAG offloading as a sequential decision process. Value-based Double DQN (Van Hasselt et al., 2016) and policy-gradient methods such as PPO (Schulman et al., 2017) and discrete soft actor-critic (Christodoulou, 2019) have been applied to edge offloading (Ke et al., 2022; Ma et al., 2024). Wang et al., (2022) cast per-task offloading as a sequence-to-sequence problem in HEFT-rank order and showed that learned policies can outperform heuristics.

Recent studies apply reinforcement learning to joint offloading and load balancing (Wu et al., 2024), cloud-edge-vehicle scheduling (Shi et al., 2023), federated or multi-agent training (Cui et al., 2023; Moon & Lim, 2022; Nie et al., 2023; Sun et al., 2025; Zhao et al., 2025), dependency-aware Internet of Vehicles (Liu et al., 2024; Pang et al., 2024), heterogeneous edge-cloud systems and pointer networks (Huang et al., 2026; Ju et al., 2023; Tang et al., 2023), multi-hop offloading (Ahmed et al., 2024), vehicular fog (Gao et al., 2023), and joint sensing-communication-computation (Z. Li et al., 2024). Table 1 compares environment, strategy, algorithm, objective, and limitation.

Among the closest DAG studies in Table 1, Pang et al. (2024) used a Double DQN policy on vehicle-edge-cloud DAGs and remain value-based only, with no policy-gradient comparison, and Ju et al. (2023) used a pointer-network scheduler that optimizes delay-constrained throughput rather than joint latency-energy quality of experience. This study keeps MEC offloading, HEFT ranking, and deep reinforcement learning, but holds the simulator and protocol fixed so that architecture, algorithm family, and stabilization can be isolated.

Table 1. Comparison of representative recent studies on reinforcement learning-based task offloading.

Reference	Year	Computing environment	Offloading strategy	RL algorithm	Optimization objective	Main limitation
Wang et al.	2022	Mobile edge computing	Dependent-task (DAG) sequence	PPO-LSTM; Double DQN	Latency-energy (QoE)	Single edge server; no architecture or tuning ablation
Wu et al.	2024	Vehicular edge computing	Offloading and load balancing	TD3	Latency-energy and load balance	No task dependency; single algorithm
Liu et al.	2024	Internet-of-Vehicle's edge	Dependency-aware (DAG) online	DQN (DAG-DQN)	Latency-energy	Single algorithm; heuristic baselines
Pang et al.	2024	Vehicle-edge-cloud	Dependency-aware (DAG)	Double DQN	Latency-energy	Value-based only; no policy-gradient comparison
Ju et al.	2023	Internet-of-Vehicles edge	Scheduling (pointer network)	DRL pointer network	Delay-constrained throughput	Single model; energy not jointly optimized
This study	2026	Mobile edge computing (single server)	Dependency-aware (DAG) sequence	LSTM/Transformer PPO, Double DQN, discrete SAC, tuned PPO	Latency-energy (QoE)	Simulated single server; fixed channel constants

To address this gap, the study keeps the simulator, task graphs, reward, and evaluation protocol fixed while comparing LSTM-PPO, Transformer-PPO, Double DQN, discrete SAC, and stabilized PPO across architecture, algorithm family, and training configuration.

METHOD

Application and Resource Model

We consider an application as a directed acyclic graph with N tasks. Each task is characterized by a processing data size and by input/output data measured in bytes; local and MEC stage times are obtained by dividing that processing size by the corresponding processing capability in Eq. (1), whereas each edge represents a precedence constraint and an associated data transfer. Every task receives a binary decision: local execution on the mobile processor or offloading to the MEC server over a wireless link. The simulator, ported without modification from (Wang et al., 2022), models two execution paths: local processing on the mobile device and an offloading path consisting of the serialized uplink, MEC execution, and downlink stages. The MEC server processing capability is ten times that of the mobile device. Stage times follow Eq. (1). Fig. 1 shows the offloading flow.

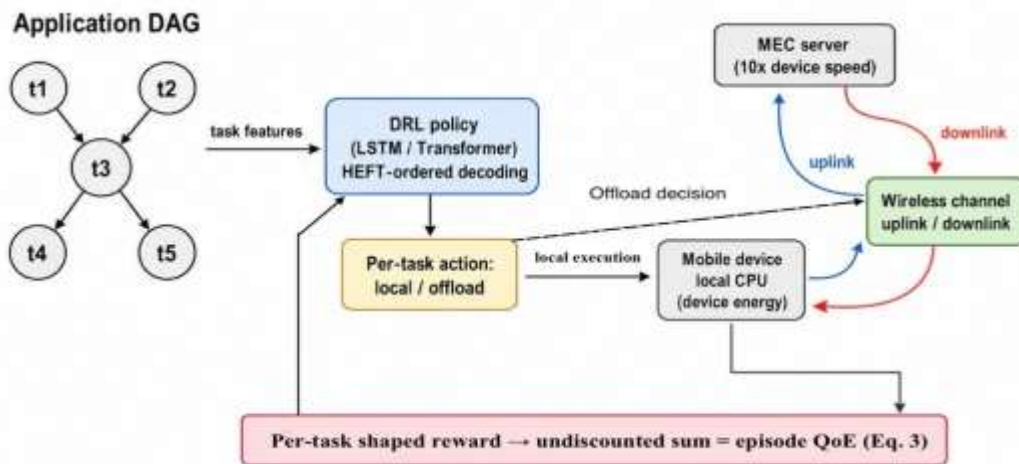


Fig. 1. Dependency-aware task offloading architecture and decision pipeline.

$$T_i^l = \frac{D_i}{C_m}, T_i^u = \frac{d_i^u}{R^u}, T_i^s = \frac{D_i}{C_s}, T_i^d = \frac{d_i^d}{R^d} \quad (1)$$

Here C_m and C_s are the mobile-device and MEC processing capabilities in Table 2. They determine stage times and are distinct from the energy-model frequency f_i . T_i^l , T_i^u , T_i^s and T_i^d are local, uplink, server and

*name of corresponding author



This is an Creative Commons License This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

downlink times; D_i is processing size; d_i^u and d_i^d are uplink and downlink data; R^u and R^d are link rates. Eq. (1) converts workload and data into simulator time so local execution can be compared with offloading. Given a complete plan, the simulator returns the application makespan (the finish time of the last exit task) and the mobile-device energy consumption, which sums local computation energy over locally executed tasks and radio transmit and receive energy over offloaded tasks, as in Eq. (2):

$$E(a) = \sum(i: a_i = 0) T_i^l \cdot \rho \cdot f_i^\zeta + \sum(i: a_i = 1) (P_{(tx)} \cdot T_i^u + P_{(rx)} \cdot T_i^d) \quad (2)$$

In Eq. (2), T_i^l is the local execution time from Eq. (1), equal to D_i/Cm and not D_i/f_i . The coefficient $\rho = 1.25 \times 10^{-26}$, the local processor frequency $f_i = 0.8 \times 10^9$, and the exponent $\zeta = 3$ appear only in the local-energy term. For an offloaded task, the device transmit and receive powers are $P_{(tx)} = 1.258$ and $P_{(rx)} = 1.181$, multiplying the uplink and downlink times T_i^u and T_i^d . Latency and energy are reported in simulator units. A task may begin a stage only when the stage resource is free and all predecessor results are present on the correct side, which forces a transfer whenever a dependency crosses the device-server boundary. In Eq. (2), $a_i = 0$ denotes local execution and $a_i = 1$ denotes offloading, while $E(a)$ is the total mobile-device energy of plan a . The first term represents local CPU energy and the second represents radio transmit/receive energy for offloaded tasks. Eq. (2) therefore provides the energy component used in the QoE trade-off in Eq. (3). All physical constants are unchanged from the original framework so that results remain directly comparable.

The energy model represents mobile-device energy consumption and does not include the energy consumed by the edge server. Consequently, the reported energy-efficiency results should be interpreted from the device perspective rather than as total system energy.

Quality of Experience Metric

Plans are scored by their normalized improvement over the all-local plan, which requires no communication and therefore serves as a natural reference. The quality of experience is the weighted, normalized reduction in latency and energy relative to that baseline, as in Eq. (3):

$$QoE(a) = -\lambda_t \cdot \frac{L(a) - L_{loc}}{L_{loc}} - \lambda_e \cdot \frac{E(a) - E_{loc}}{E_{loc}} \quad (3)$$

$L(a)$ and $E(a)$ are the makespan and mobile-device energy of plan a ; $L_{(loc)}$ and $E_{(loc)}$ are the all-local references. λ_t and λ_e are the latency and energy weights; the latency-oriented goal uses $(\lambda_t, \lambda_e) = (1, 0)$ and the energy-aware goal uses $(0.5, 0.5)$. Eq. (3) measures normalized improvement over all-local execution; positive values indicate improvement. In our evaluation we report the latency, energy, quality of experience and convergence speed because they capture responsiveness, battery cost, the overall trade-off for the users and how quickly a method becomes usable after training begins.

Reinforcement Learning Formulation

Offloading one task graph is one episode. Tasks are first sorted by HEFT upward rank, and the agent emits one decision per task in that order; because resource availability depends on all earlier decisions, the process is inherently sequential. The state is a 17-dimensional vector of the current HEFT-ordered task, with this fixed order: (1) original task index; (2–5) unnormalized local, uplink, MEC, and downlink stage times; (6–11) up to six predecessor original indices; (12–17) up to six successor original indices. Empty slots are -1; extra neighbors beyond six are dropped in increasing original-index order; indices are not scaled. This 17-dimensional vector is the complete per-task input provided to the agent; no additional input features are used. The action is binary. The per-task reward is $r_t = \lambda_t \cdot (-\Delta L_t - L_{loc}/N)/L_{loc} + \lambda_e \cdot (-\Delta E_t - E_{loc}/N)/E_{loc}$; summing over t recovers Eq. (3). Training uses discount 0.99, so the return is a discounted surrogate of that sum. The undiscounted sum of per-task rewards equals the episode QoE in Eq. (3); the discount is used only during training and is not part of Eq. (3). Here ΔL_t and ΔE_t are the latency and mobile-device energy increments of task t , and N is the number of tasks in the graph.

Baseline and Evaluated Configurations

The first baseline reproduces the original policy: an input projection, a two-layer bidirectional LSTM encoder (Hochreiter & Schmidhuber, 1997), and an autoregressive decoder that emits one action per task conditioned on the previous action and the encoder memory. It is trained with PPO using the clipped surrogate objective of Eq. (4), which limits the policy update; the advantage comes from generalized advantage estimation, and a clipped value loss and entropy bonus aid stability and exploration.

$$L^{CLIP}(\theta) = E_t[\min(r_t(\theta) \cdot \hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon) \cdot \hat{A}_t)] \quad (4)$$

*name of corresponding author



This is an Creative Commons License This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

Here $r_t(\theta)$ is the probability ratio of the new and old policies, $\hat{A}_t$ is the advantage, and ε is the clip range. Eq. (4) limits how far a PPO update may move the policy in one step.

The second baseline is a Double DQN agent whose bidirectional-LSTM Q-network outputs an action value at each task position; decisions are transitions along the prioritized sequence, and the target selects the action with the online network but evaluates it with a periodically updated target network to reduce overestimation.

The first evaluated configuration replaces the recurrent encoder-decoder with a Transformer (Y.-S. Li & Gau, 2024) while keeping the PPO trainer identical. It uses an input projection, sinusoidal positional encoding, a two-layer self-attention encoder, and an autoregressive decoder with causal masking and cross-attention; self-attention lets every task attend to all others in one step.

The second evaluated configuration is a discrete soft actor-critic agent that introduces off-policy maximum-entropy learning. A context encoder feeds a categorical actor and twin critics with slowly updated targets. Because the action space is discrete, the soft state value is computed exactly by summing over the two actions, as in Eq. (5):

$$V(s) = \sum_a \pi(a|s) \cdot (\min_k Q_k(s, a) - \alpha \cdot \log \pi(a|s)) \quad (5)$$

Here $\pi(a | s)$ is the action probability, Q_k are the twin critics, and α is the temperature. Eq. (5) is the soft state value used to train discrete SAC. Its critic target bootstraps to the next task, the temperature is tuned toward a target entropy, and the replay buffer stores whole episodes bucketed by task count. The third evaluated configuration, stabilized PPO, enables modern stabilization in the same trainer through configuration: per-batch advantage normalization, linear annealing of the learning rate, clip range, and entropy coefficient, Kullback-Leibler early stopping, and orthogonal initialization. Because the baseline PPO-LSTM uses four optimization epochs per update whereas stabilized PPO uses six, a six-epoch PPO-LSTM control was added under the same environment-interaction budget. The control changes only the number of PPO epochs, allowing the effect of additional optimization to be separated from the stabilization components. Baseline PPO-LSTM used 40 optimizer steps per update, compared with 60 for PPO-LSTM-6E and an average of 59.916 for stabilized PPO because of occasional KL-based early stopping.

Five non-learning policies—all-local, all-MEC, round-robin, uniform random, and HEFT upward-rank greedy (Topcuoglu et al., 2002)—are scored through the identical path for reference.

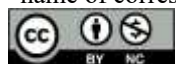
Experimental Setup

Task graphs are generated with DAGGEN for sizes 10, 15, ..., 50. Each graph uses $\text{fat} \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$, $\text{density} \in \{0.5, 0.6, 0.7, 0.9\}$, $\text{regularity} \in \{0.5, 0.7, 0.9\}$, $\text{communication-to-computation ratio} \in \{0.3, 0.4, 0.5\}$, and stored DAGGEN header parameters --mindata 5242880 and --maxdata 52428800. Because the original per-graph generator seed and DAGGEN binary version were not retained, the stored DAG graph files define the dataset used in the experiments. Training uses indices 0–499 for bandwidth experiments and 0–299 for task-size and ablation experiments; validation uses 800–899, and the independent final test uses 900–999 once after training. Legacy test folders duplicating training graphs 0–99 were not used. The task-size analysis trains on the mixed-size set and evaluates per size at fixed bandwidth; the bandwidth analysis fixes 15-task graphs and varies 3, 5, 7, 9, and 11 Mbps. Each analysis uses both goals and five random seeds, with shared splits. Comparisons pair five training seeds with 100 final-test graphs. Primary inference uses a crossed bootstrap ($B = 10,000$) with 95% confidence intervals and Holm–Bonferroni correction; seed-level median differences, rank-biserial effects, positive/negative/tied counts, and an exact sign-flip test are also reported. The 100 final-test graphs are evaluation instances, not independent replicates; the previous $n = 100$ Wilcoxon analysis is not used. Convergence is measured on validation QoE with a common threshold, not each method’s own peak. For each comparable cell, $\tau = 0.90$ times the best learned method’s seed-mean validation QoE at update 150. A seed converges at the first logged update with validation QoE $\geq \tau$; if it never reaches τ the value is unreported (not replaced by 150). We also report the normalized area under the validation learning curve. Final-test graphs are not used to compute τ or convergence.

Table 2. System, simulation, and algorithm parameters extracted from the source code. Energy is mobile-device energy only; latency and energy are reported in simulator units.

Parameter	Value / setting
Mobile / MEC processing capability (C_m/C_s)	$1.05 \times 10^6 / 1.05 \times 10^7$ units/s (MEC = $10 \times$ device)
Wireless bandwidth & up/downlink rate	3, 5, 7, 9, 11 Mbps (default 7); rate = bandwidth $\times$ 1024 $\times$ 1024/8 bytes/s (symmetric)
Transmit power $P_{(tx)}$	1.258 energy units per unit uplink time T_i^u
Receive power $P_{(rx)}$	1.181 energy units per unit downlink time T_i^d
Local execution time T_i^l	$T_i^l = D_i / C_m$ from Eq. (1)
Local processor frequency f_l	0.8×10^9 ; used only in Eq. (2), not in stage times

*name of corresponding author



This is anCreative Commons License This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

Parameter	Value / setting
Local computation energy	$E_i^l = \rho(f_i)^\zeta T_i^l$; $\rho = 1.25 \times 10^{-26}$, $\zeta=3$
Task-graph generator / state	DAGGEN; sizes 10, 15, ..., 50; fat $\in \{0.1, 0.3, 0.5, 0.7, 0.9\}$; density $\in \{0.5, 0.6, 0.7, 0.9\}$; regularity $\in \{0.5, 0.7, 0.9\}$; CCR $\in \{0.3, 0.4, 0.5\}$; stored DAGGEN header parameters -mindata 5242880 and --maxdata 52428800. State (17): index, local/uplink/MEC/downlink times (unnormalized), 6 predecessor ids, 6 successor ids; pad -1; truncate extras; ids not scaled
Objective weights (λ_t, λ_e)	latency-oriented (1, 0); energy-aware (0.5, 0.5)
Latency / energy units	simulator time / energy units; energy is mobile-device only
Training / validation / final-test graphs	Bandwidth: 500 train (indices 0–499), 100 validations (800–899), 100 final-test (900–999) on 15-task graphs. Task-size and ablation: 300 train (0–299), 100 validations (800–899), 100 final-test (900–999) per size. Previously labelled test folders are unused.
Seeds / updates / episodes	5 seeds (0–4); 150 updates; 10 episodes per update
Discount γ / GAE λ	0.99 / 0.95
Learning rate	PPO 5×10^{-4} ; SAC 3×10^{-4} ; Double DQN 5×10^{-4}
PPO settings	minibatch 500; epochs 4 (baseline) / 6 (stabilized and six-epoch control); clip 0.2; entropy 0.01. The six-epoch control uses baseline settings except the epoch count
Double DQN	20 steps; batch 64 episodes; replay 20,000; hard target every 5 updates
Discrete SAC	20 steps; batch 64; replay 20,000; Polyak 0.01; target entropy $0.7 \cdot \log 2$

Experiments were executed on Windows 11 using Python 3.12, PyTorch 2.6, and an NVIDIA RTX 4060 Laptop GPU.

RESULTS

Results are reported for task-graph size, bandwidth, training curves, ablation, and seed-aware comparisons. QoE is the primary metric; learning curves use validation QoE, whereas tables report one-shot final-test scores. Tables 3 and 4 report the five learners and HEFT; remaining fixed policies appear in Figures 2 and 3. Table 3 and Fig. 2 report performance for graphs of ten to fifty tasks.

The PPO-LSTM configurations remained closely matched over the evaluated task-graph sizes. Averaged over sizes, stabilized PPO achieved a mean QoE of 0.389 for the latency-oriented objective and 0.426 for the energy-aware objective, compared with 0.388 and 0.426 for baseline PPO-LSTM and 0.385 and 0.422 for Transformer-PPO, respectively. Against HEFT means of 0.323 and 0.276, the corresponding absolute QoE gains were 0.066 and 0.150. Thus, stabilized PPO was marginally higher than baseline PPO-LSTM for the latency-oriented objective, whereas baseline PPO-LSTM was marginally higher for the energy-aware objective. Double DQN and discrete soft actor-critic trail on the latency-oriented objective, where soft actor-critic falls below HEFT.

Table 3. Performance across task-graph sizes (10–50 tasks). QoE, latency, and mobile-device energy are averaged across task sizes; Conv. is the median update reaching the common validation threshold. A dash indicates that the threshold was not reached.

Goal	Method	QoE@10	QoE@50	QoE (mean $\pm$ SD)	Latency	Energy	Conv.
Latency-oriented	Stabilized PPO	0.282	0.438	0.389 ± 0.062	1266	9439	5 (5/5)
	Baseline PPO-LSTM	0.281	0.437	0.388 ± 0.062	1266	9469	5 (5/5)
	Transformer-PPO	0.277	0.436	0.385 ± 0.062	1273	9411	10 (5/5)
	Double DQN	0.202	0.202	0.200 ± 0.018	1724	10993	—
	Discrete SAC	0.084	0.240	0.186 ± 0.060	1704	11341	—
	HEFT	0.249	0.352	0.323 ± 0.041	1424	10463	—
Energy-aware	Stabilized PPO	0.372	0.452	0.426 ± 0.032	1400	6508	5 (5/5)

*name of corresponding author



This is anCreative Commons License This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

Goal	Method	QoE@10	QoE@50	QoE (mean $\pm$ SD)	Latency	Energy	Conv.
	Baseline PPO-LSTM	0.372	0.451	0.426 $\pm$ 0.032	1399	6513	5 (5/5)
	Transformer-PPO	0.369	0.447	0.422 $\pm$ 0.031	1453	6286	10 (5/5)
	Double DQN	0.319	0.379	0.357 $\pm$ 0.025	1836	5654	35 (2/5)
	Discrete SAC	0.348	0.386	0.375 $\pm$ 0.020	1908	4783	95 (3/5)
	HEFT	0.213	0.299	0.276 $\pm$ 0.034	1424	10463	—

When averaged over the five bandwidths, stabilized PPO averaged QoE 0.308 (latency-oriented) and 0.358 (energy-aware), versus HEFT means of 0.261 and 0.222, corresponding to absolute gains of 0.047 and 0.136. At 3 Mbps, learned policies retained selective offloading whereas the all-MEC policy degraded sharply. Detailed latency, energy, and offloading results are reported in Table 4.

Table 4. Results across wireless bandwidth levels (fixed 15-task graphs). QoE at 3 and 11 Mbps and the mean $\pm$ SD across the five levels (higher is better); latency and mobile-device energy are means across levels (lower is better); offload is the mean offloading ratio; Conv. is the median update to the common validation threshold at the default 7 Mbps; each bandwidth has its own τ . Best learned QoE per goal in bold.

Goal	Method	QoE@3	QoE@11	QoE (mean $\pm$ SD)	Latency	Energy	Offload	Conv.
Latency-oriented	Stabilized PPO	0.164	0.460	0.308 $\pm$ 0.113	719	4416	0.53	15 (5/5)
	Baseline PPO-LSTM	0.163	0.458	0.306 $\pm$ 0.113	721	4417	0.53	20 (5/5)
	Transformer-PPO	0.155	0.454	0.298 $\pm$ 0.114	729	4382	0.54	20 (5/5)
	Double DQN	0.117	0.345	0.237 $\pm$ 0.086	794	4865	0.43	—
	Discrete SAC	0.039	0.327	0.180 $\pm$ 0.113	856	4981	0.40	—
	HEFT	0.137	0.351	0.261 $\pm$ 0.086	770	5456	—	—
Energy-aware	Stabilized PPO	0.113	0.586	0.358 $\pm$ 0.195	786	3524	0.71	1 (5/5)
	Baseline PPO-LSTM	0.114	0.586	0.359 $\pm$ 0.194	787	3514	0.71	1 (5/5)
	Transformer-PPO	0.105	0.586	0.354 $\pm$ 0.198	708	3448	0.72	1 (5/5)
	Double DQN	0.086	0.584	0.334 $\pm$ 0.210	845	3475	0.70	1 (5/5)
	Discrete SAC	0.040	0.585	0.320 $\pm$ 0.228	876	3466	0.69	1 (5/5)
	HEFT	0.089	0.322	0.222 $\pm$ 0.093	770	5455	—	—

An out-of-distribution transfer test trained policies at 7 Mbps and evaluated frozen weights at 3 and 11 Mbps. Stabilized PPO remained positive at 11 Mbps (0.34 latency-oriented; 0.56 energy-aware), whereas PPO-family QoE became negative at 3 Mbps (about -0.66 and -0.56), indicating poor transfer to a much slower link. These out-of-distribution scores are not directly comparable with the in-distribution results in Table 4.

Fig. 4 shows bandwidth-averaged validation curves; mixed-size convergence is reported in Table 3, not in Fig. 4. Under the common threshold, baseline PPO-LSTM and stabilized PPO both reach τ at median update 5 on mixed-size training, and the Transformer policy at update 10. Double DQN and discrete SAC do not reach the latency-oriented common threshold in any seed. Normalized validation AUC is essentially the same for baseline and stabilized PPO (about 0.371 vs 0.372 on the latency-oriented objective). Stabilized PPO is therefore not uniquely the fastest method once all learners share the same target.

Table 5 and Fig. 5 report a leave-one-component-out ablation of the stabilization recipe on thirty-task graphs at 7 Mbps with five seeds. Paired differences used a crossed bootstrap of five training seeds and 100 final-test graphs ($B = 10\,000$), with Holm correction over twelve comparisons (six components $\times$ two objectives). On the

latency-oriented objective, removing advantage normalization did not reduce mean quality of experience ($\Delta\text{QoE} = +0.0042$; Holm-adjusted $p = 0.23$). The largest latency-oriented drop was from removing orthogonal initialization ($\Delta\text{QoE} = -0.0020$), and that 95% confidence interval included zero. On the energy-aware objective, removing advantage normalization produced the largest drop ($\Delta\text{QoE} = -0.0035$; 95% CI $(-0.0056, -0.0014)$; Holm-adjusted $p = 0.019$; 5/5 seeds; rank-biserial $r = 1.00$) and was the only Holm-significant leave-one-out comparison. Leave-one-out removal does not establish that any component independently caused a bundled gain. Disabling Kullback–Leibler early stopping left final-test quality unchanged (5/5 ties). Removing advantage normalization slowed latency-oriented convergence from 15 to 35 updates; every energy-aware variant reached τ at update 5.

A six-epoch PPO-LSTM control on the same 30-task, 7 Mbps cell did not improve mean QoE relative to the four-epoch baseline (latency-oriented: 0.4255 baseline vs 0.4247 six-epoch; energy-aware: 0.4463 vs 0.4440). The full recipe was not Holm-significant versus either control. Matching compute by adding two epochs is therefore not sufficient to account for differences among these PPO variants.

Table 5. PPO stabilization ablation on 30-task graphs at 7 Mbps (leave-one-component-out over five seeds). $\Delta\text{QoE} = \text{variant} - \text{full stabilized PPO}$ for the latency-oriented (LO) and energy-aware (EE) objectives; higher values indicate better performance relative to the full configuration. Conv. is the median update to the common validation threshold. Training-time comparisons are reported in Table 6.

Variant	$\Delta\text{QoE-LO}$	$\Delta\text{QoE-EE}$	Conv.-LO	Conv.-EE
Baseline PPO-LSTM	-0.0006	+0.0014	20 (5/5)	5 (5/5)
PPO-LSTM-6E	-0.0014	-0.0010	15 (5/5)	5 (5/5)
Full stabilized PPO	0 (ref)	0 (ref)	15 (5/5)	5 (5/5)
– advantage normalization	+0.0042	-0.0035	35 (5/5)	5 (5/5)
– learning-rate annealing	+0.0016	+0.0020	15 (5/5)	5 (5/5)
– clip-range annealing	+0.0009	+0.0004	15 (5/5)	5 (5/5)
– entropy annealing	+0.0000	+0.0006	15 (5/5)	5 (5/5)
– KL early stopping	0.0000	0.0000	15 (5/5)	5 (5/5)
– orthogonal initialization	-0.0020	+0.0001	15 (5/5)	5 (5/5)

After crossed-bootstrap inference with Holm–Bonferroni correction, stabilized PPO did not differ significantly from either baseline PPO-LSTM or the six-epoch PPO-LSTM control at any evaluated bandwidth under either objective. Compared with Transformer-PPO, a significant difference remained only at 5 Mbps for the latency-oriented objective. Compared with HEFT, stabilized PPO was significantly higher at 9 and 11 Mbps for the latency-oriented objective and at all five bandwidths for the energy-aware objective. Stabilized PPO matched baseline PPO-LSTM in trainable parameters and inference time but required greater training time per update.

Table 6. Summary of seed-aware QoE comparisons across five bandwidths and computational overhead. Panel A reports the key Holm-corrected findings; complete bandwidth-specific statistics are provided in Supplementary Table S1. Panel B reports computational overhead under identical hardware conditions.

Panel A. Key seed-aware QoE findings				
Comparator	Goal	Holm-significant bandwidth(s)	Main finding	
Baseline PPO-LSTM	Latency-oriented	None	No significant difference	
PPO-LSTM-6E	Latency-oriented	None	No significant difference	
Transformer-PPO	Latency-oriented	5 Mbps	Stabilized PPO higher	
HEFT	Latency-oriented	9, 11 Mbps	Stabilized PPO higher	
Baseline PPO-LSTM	Energy-aware	None	No significant difference	
PPO-LSTM-6E	Energy-aware	None	No significant difference	
Transformer-PPO	Energy-aware	None	No significant difference	
HEFT	Energy-aware	3, 5, 7, 9, 11 Mbps	Stabilized PPO higher	
Panel B. Computational overhead (30-task graphs, identical hardware): trainable parameters, training time per update, inference time per graph, and peak GPU memory				
Method	Trainable parameters	Training time (s/update)	Inference time (ms)	Peak GPU memory (MB)

*name of corresponding author



This is anCreative Commons License This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

Baseline PPO-LSTM	1,355,107	3.59	0.30	376
Stabilized PPO	1,355,107	4.94	0.29	386
Transformer-PPO	685,923	4.78	1.08	366
Double DQN	201,218	0.50	0.02	196
Discrete SAC	201,218	0.72	0.02	199

Note. Complete mean and median paired differences, 95% confidence intervals, Holm-corrected bootstrap and exact p-values, rank-biserial effect sizes, and seed-level positive/negative/tied counts are reported in Supplementary Table S1. PPO-LSTM-6E denotes the six-optimization-epoch PPO-LSTM control.

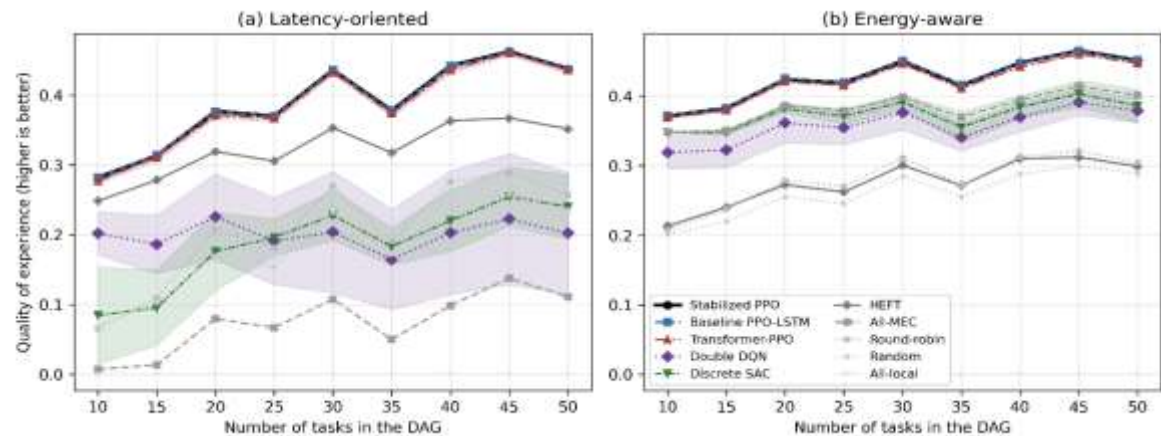


Fig. 2. QoE versus task-graph size for the latency-oriented (a) and energy-aware (b) objectives; learned-method means with 95% confidence bands over five seeds, heuristics deterministic. Higher is better.

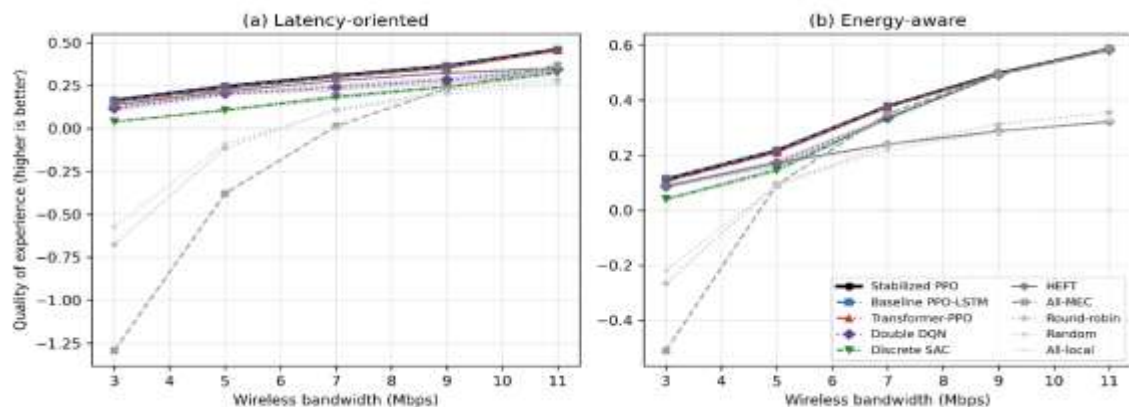


Fig. 3. QoE versus wireless bandwidth (fixed 15-task graphs) for the latency-oriented (a) and energy-aware (b) objectives; 95% confidence bands over five seeds. Higher is better.

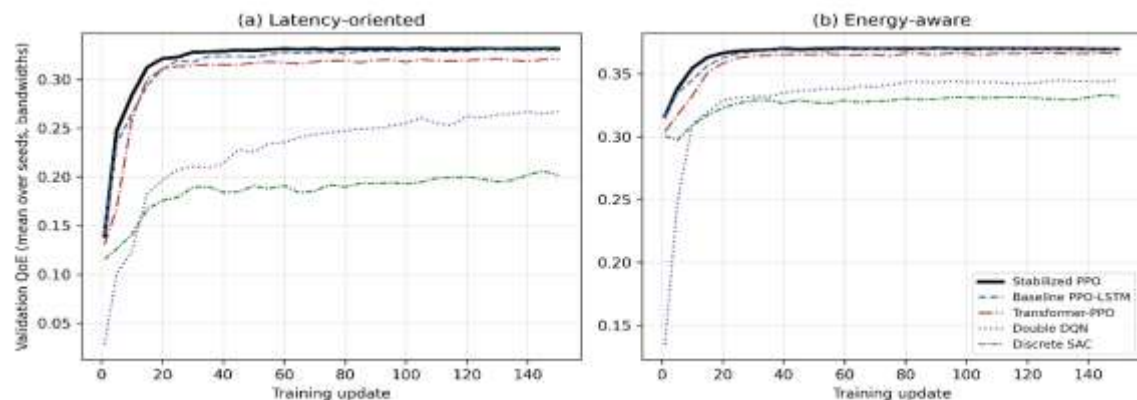


Fig. 4. Training curves: validation QoE versus update, averaged over seeds and bandwidths, for the latency-oriented (a) and energy-aware (b) objectives.

*name of corresponding author



This is anCreative Commons License This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

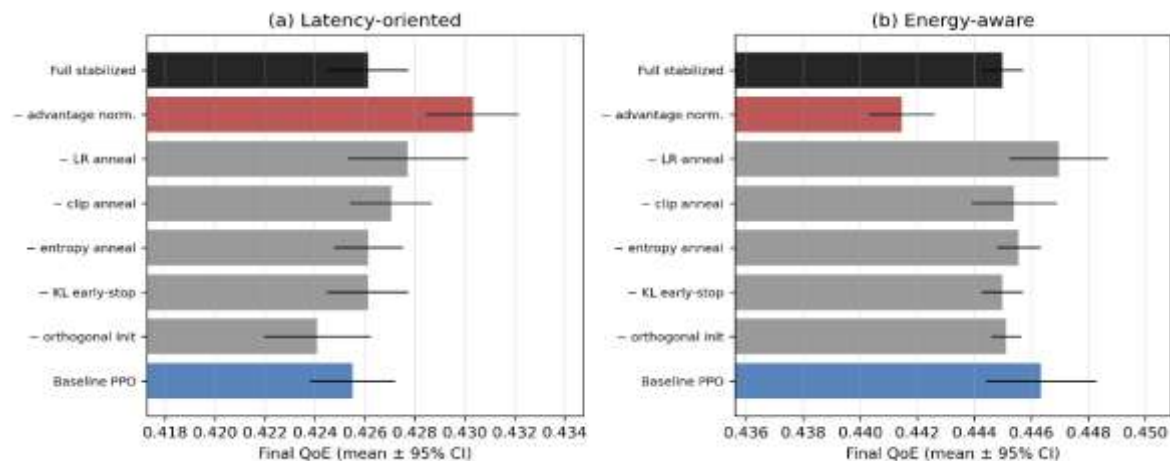


Fig. 5. PPO stabilization ablation (30-task graphs, 7 Mbps): final QoE (mean and 95% CI over five seeds) for the full configuration, each leave-one-component-out variant, and the baseline. LR denotes learning rate, and KL denotes Kullback–Leibler divergence.

DISCUSSION

Performance and Robustness

In both analyses, the PPO-LSTM configurations achieved the strongest overall QoE and remained closely matched. The seed-aware difference between stabilized PPO and baseline PPO-LSTM was not Holm-significant at any bandwidth, and both reached τ at the same median update, so faster convergence cannot be attributed uniquely to stabilization. Double DQN and discrete SAC were slower or did not reach the latency-oriented threshold. Leave-one-component-out ablation found a Holm-significant quality drop only when advantage normalization was removed under the energy-aware objective ($\Delta\text{QoE} = -0.0035$), and that drop was small; it does not establish an independent causal effect. Ranking was stable from ten to fifty tasks. Quality of experience rose with bandwidth as learned policies adapted their offloading ratio; the all-MEC policy collapsed at 3 Mbps, so adaptive placement was most valuable under constrained links.

The main contribution is the controlled comparison of evaluated configurations, not a new algorithm. Under that pipeline the gain over PPO-LSTM remains small; alternative architectures and learning families did not yield a larger quality improvement in the tested setting.

Algorithmic and Computational Trade-offs

The Transformer-PPO configuration was competitive but did not outperform the recurrent policy, staying within about 0.003 (0.385 versus 0.388 on the task-size latency-oriented setting). Short sequences, encoded dependency features, and slower decoding under an equal budget may explain this; deeper graphs with graph-aware attention may favor attention-based policies (Ju et al., 2023; Z. Li et al., 2024). Discrete SAC reached the lowest device energy among learned methods under the energy-aware objective, about 4783 versus 6508 mean device-energy units, below stabilized PPO, but was the weakest learner under pure latency optimization and fell below HEFT, consistent with delayed dependence in bootstrapped off-policy targets (Christodoulou, 2019). All learners decide tasks in the same HEFT-rank order and observe at most six padded neighbors, so architecture differences are partly absorbed by a shared sequential bias; that encoding also limits transfer when the link changes. Advantage normalization rescales per-task shaped rewards whose magnitude varies with graph size, which is consistent with its larger effect when latency and energy are jointly weighted. These findings apply to a single-user, single-server simulator. Stabilized PPO matched the baseline in model size and inference latency but required more training time; Transformer-PPO had the slowest inference.

Comparison, Limitations, and Implications

Consistent with (Wang et al., 2022) and dependency-aware offloading for the Internet of Vehicles (Liu et al., 2024; Pang et al., 2024), PPO-based policies outperformed HEFT, whereas Double DQN and discrete SAC depended on the objective. The slower convergence of Double DQN mirrored value-based bootstrapping in vehicular reinforcement learning (Shi et al., 2023; Zhou et al., 2025), and the energy-latency trade-off aligned with joint delay-energy studies (Y.-S. Li & Gau, 2024; Wu et al., 2024). Unlike distributed or federated approaches (Cui et al., 2023; Moon & Lim, 2022; Nie et al., 2023; Sun et al., 2025; Zhao et al., 2025), this study isolated architecture, algorithm family, and tuning under one pipeline. Limitations include a simulated single-server system with fixed channel constants, no mobility or interference, a device-only energy model, graphs of ten to fifty tasks,

*name of corresponding author



This is anCreative Commons License This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

a single generator, greedy test decoding, and a 150-update budget. Matching the recurrent baseline in model size and inference time makes the stabilized configuration usable on resource-constrained mobile devices, provided the deployed bandwidth remains close to the training link.

CONCLUSION

This study presented a controlled comparison of reinforcement learning methods for dependency-aware task offloading in mobile edge computing under identical conditions. The contribution isolates architecture, learning family, and PPO stabilization under one DAG-MEC protocol. Stabilized PPO remained competitive with PPO-LSTM, but its QoE advantage was small and not Holm-significant at any evaluated bandwidth after seed-aware inference. Under a common validation threshold both PPO variants reached τ at the same median update. No single stabilization component drove the quality difference on both objectives. Transformer-PPO remained competitive, and discrete SAC achieved lower device energy under the energy-aware objective. Matching the recurrent baseline in model size and inference time suggests that stabilized PPO may be suitable for resource-constrained deployment, while substantial degradation in unseen bandwidth may require retraining or online adaptation. Future work will address dynamic channels, multiple edge servers, larger graphs, graph-aware policies, and online adaptation. The simulator, training scripts, configurations, graph files, raw per-seed results, and statistical analysis scripts are available in an anonymous public repository: <https://github.com/mec-offloading-review/dependency-aware-task-offloading-review>.

REFERENCES

- Ahmed, M., Raza, S., Ahmad, H., Khan, W. U., Xu, F., & Rabie, K. (2024). Deep reinforcement learning approach for multi-hop task offloading in vehicular edge computing. *Engineering Science and Technology, an International Journal*, 59. <https://doi.org/10.1016/j.jestch.2024.101854>
- Christodoulou, P. (2019). *Soft Actor-Critic for Discrete Action Settings*. <http://arxiv.org/abs/1910.07207>
- Cui, Y., Li, H., Zhang, D., Zhu, A., Li, Y., & Qiang, H. (2023). Multiagent Reinforcement Learning-Based Cooperative Multitype Task Offloading Strategy for Internet of Vehicles in B5G/6G Network. *IEEE Internet of Things Journal*, 10(14), 12248–12260. <https://doi.org/10.1109/JIOT.2023.3245721>
- Gao, Z., Yang, L., & Dai, Y. (2023). Fast Adaptive Task Offloading and Resource Allocation via Multiagent Reinforcement Learning in Heterogeneous Vehicular Fog Computing. *IEEE Internet of Things Journal*, 10(8), 6818–6835. <https://doi.org/10.1109/JIOT.2022.3228246>
- Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 9(8), 1735–1780. <https://doi.org/10.1162/neco.1997.9.8.1735>
- Hu, X., & Huang, Y. (2022). Deep reinforcement learning based offloading decision algorithm for vehicular edge computing. *PeerJ Computer Science*, 8. <https://doi.org/10.7717/PEERJ-CS.1126>
- Huang, J., Xiao, H., Xiao, L., & Gu, Z. (2026). Deep Reinforcement Learning-Based DAG Task Scheduling for Heterogeneous Edge-Cloud Systems. *Journal of Circuits, Systems and Computers*. <https://doi.org/10.1142/S0218126626502361>
- Ju, X., Su, S., Xu, C., & Wang, H. (2023). Computation offloading and tasks scheduling for the internet of vehicles in edge computing: A deep reinforcement learning-based pointer network approach. *Computer Networks*, 223, 109572. <https://doi.org/10.1016/j.comnet.2023.109572>
- Ke, H., Wang, H., & Sun, H. (2022). Multi-Agent Deep Reinforcement Learning-Based Partial Task Offloading and Resource Allocation in Edge Computing Environment. *Electronics*, 11(15), 2394. <https://doi.org/10.3390/electronics11152394>
- Li, Y.-S., & Gau, R.-H. (2024). Transformer-Assisted Deep Reinforcement Learning for Distributed Latency-Sensitive Task Offloading in Mobile Edge Computing. *ICC 2024 - IEEE International Conference on Communications*, 2944–2949. <https://doi.org/10.1109/ICC51166.2024.10622508>
- Li, Z., Fu, Y., Tian, M., & Li, C. (2024). Cooperative sensing, communication and computation resource allocation in mobile edge computing-enabled vehicular networks. *Journal of Information and Intelligence*, 2(4), 339–354. <https://doi.org/10.1016/j.jiixd.2024.02.006>
- Liu, C., Wang, H., Zhao, M., Liu, J., Zhao, X., & Yuan, P. (2024). Dependency-aware online task offloading based on deep reinforcement learning for IoV. *Journal of Cloud Computing*, 13(1). <https://doi.org/10.1186/s13677-024-00701-0>
- Ma, B., Xu, Y., Pan, Y., Liu, S., & Li, C. (2024). A multi-user mobile edge computing task offloading and trajectory management based on proximal policy optimization. *Peer-to-Peer Networking and Applications*, 17(6), 4210–4229. <https://doi.org/10.1007/s12083-024-01796-7>
- Moon, S., & Lim, Y. (2022). Federated Deep Reinforcement Learning Based Task Offloading with Power Control in Vehicular Edge Computing. *Sensors*, 22(24), 9595. <https://doi.org/10.3390/s22249595>

*name of corresponding author



This is anCreative Commons License This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

- Nie, X., Yan, Y., Zhou, T., Chen, X., & Zhang, D. (2023). A Delay-Optimal Task Scheduling Strategy for Vehicle Edge Computing Based on the Multi-Agent Deep Reinforcement Learning Approach. *Electronics*, 12(7), 1655. <https://doi.org/10.3390/electronics12071655>
- Pang, S., Hou, L., Gui, H., He, X., Wang, T., & Zhao, Y. (2024). Multi-mobile vehicles task offloading for vehicle-edge-cloud collaboration: A dependency-aware and deep reinforcement learning approach. *Computer Communications*, 213, 359–371. <https://doi.org/10.1016/j.comcom.2023.11.013>
- Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). *Proximal Policy Optimization Algorithms*. <http://arxiv.org/abs/1707.06347>
- Shi, W., Chen, L., & Zhu, X. (2023). Task Offloading Decision-Making Algorithm for Vehicular Edge Computing: A Deep-Reinforcement-Learning-Based Approach. *Sensors*, 23(17), 7595. <https://doi.org/10.3390/s23177595>
- Sivasakthi, D. A., & Gunasekaran, R. (2022). QoE-aware mobile computation offloading in mobile edge computing. *Concurrency and Computation: Practice and Experience*, 34(11). <https://doi.org/10.1002/cpe.6853>
- Sun, X., Hu, Y., Gao, X., & Wang, H. (2025). A cooperative multi-agent optimization approach for task offloading in vehicular edge computing systems. *Discover Computing*, 28(1). <https://doi.org/10.1007/s10791-025-09887-6>
- Tang, T., Li, C., & Liu, F. (2023). Collaborative cloud-edge-end task offloading with task dependency based on deep reinforcement learning. *Computer Communications*, 209, 78–90. <https://doi.org/10.1016/j.comcom.2023.06.021>
- Topcuoglu, H., Hariri, S., & Wu, M.-Y. (2002). Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems*, 13(3), 260–274. <https://doi.org/10.1109/71.993206>
- Uddin, A., Sakr, A. H., & Zhang, N. (2024). Prioritized Task Offloading in Vehicular Edge Computing Using Deep Reinforcement Learning. *2024 IEEE 99th Vehicular Technology Conference (VTC2024-Spring)*, 1–6. <https://doi.org/10.1109/VTC2024-Spring62846.2024.10683050>
- Van Hasselt, H., Guez, A., & Silver, D. (2016). Deep reinforcement learning with double Q-learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 30(1). <https://doi.org/10.1609/aaai.v30i1.10295>
- Wang, J., Hu, J., Min, G., Zhan, W., Zomaya, A. Y., & Georgalas, N. (2022). Dependent task offloading for edge computing based on deep reinforcement learning. *IEEE Transactions on Computers*, 71(10), 2449–2461. <https://doi.org/10.1109/tc.2021.3131040>
- Wu, Z., Jia, Z., Pang, X., & Zhao, S. (2024). Deep Reinforcement Learning-Based Task Offloading and Load Balancing for Vehicular Edge Computing. *Electronics*, 13(8), 1511. <https://doi.org/10.3390/electronics13081511>
- Yang, W., Liu, Z., Liu, X., & Ma, Y. (2024). Deep reinforcement learning-based low-latency task offloading for mobile-edge computing networks. *Applied Soft Computing*, 166, 112164. <https://doi.org/10.1016/j.asoc.2024.112164>
- Zhang, Y., Chen, J., Zhou, Y., Yang, L., He, B., & Yang, Y. (2022). Dependent task offloading with energy-latency tradeoff in mobile edge computing. *IET Communications*, 16(17), 1993–2001. <https://doi.org/10.1049/cmu2.12454>
- Zhao, H., Li, Y., Pang, Z., & Ma, Z. (2025). Federated Multi-Agent DRL for Task Offloading in Vehicular Edge Computing. *Electronics*, 14(17), 3501. <https://doi.org/10.3390/electronics14173501>
- Zhou, J., Liang, J., Zhao, L., Wan, S., Cai, H., & Xiao, F. (2025). Latency-Energy Efficient Task Offloading in the Satellite Network-Assisted Edge Computing via Deep Reinforcement Learning. *IEEE Transactions on Mobile Computing*, 24(4), 2644–2659. <https://doi.org/10.1109/TMC.2024.3502643>