A Decentralized Multi-Agent Reinforcement Learning Framework for Joint Task Offloading and Service Caching in Fog Computing: A Methodological Approach

Kaushik Mishra[1], Ganesh Khekare[2]

[1]Lincoln University College (LUC), 47301, Petaling Jaya, Selangor Darul Ehsan, Malaysia

[2]School of Computer Science and Engineering, Vellore Institute of Technology, Vellore, 632014, Tamil Nadu, India

E-mail ID: pdf.kaushik@lincoln.edu.my, khekare.123@gmail.com

Abstract: The rapid proliferation of Internet of Things (IoT) devices and latency-sensitive applications has significantly increased computational demands at the network edge. Traditional cloud-centric architecture often fails to meet strict delay requirements due to centralized processing and network congestion. Mobile Edge Computing (MEC) addresses this challenge by deploying computing and storage resources closer to end users. However, efficiently managing limited edge resources under dynamic workloads remains a major challenge, particularly when coordinating task offloading and service caching decisions. This work proposes a decentralized multi-agent reinforcement learning (MARL) framework for joint task offloading and service caching in a cloud-assisted MEC environment. Each MEC-enabled base station operates as an autonomous learning agent that observes local system states and dynamically determines optimal offloading and caching actions. The decision process is modeled as a decentralized partially observable Markov decision process to enable scalable distributed learning. The proposed framework aims to minimize overall operational cost by reducing latency, energy consumption, and cache miss rates while improving resource utilization across fog nodes.

Keywords: Mobile Edge Computing; Service caching; Service Replacement; Multi-agent reinforcement learning Task offloading; Cloud computing

---

Introduction

The rapid growth of IoT devices and delay-critical applications has caused an unpredicted surge in data generation at the network edge. These traditional cloud-centric paradigms often fail to meet these requirements due to traffic overhead, prolonged transmission delays, and limited adaptability to variable network conditions [1]. Mobile Edge Computing (MEC) [2] has become a promising solution by putting computing and storage resources closer to the people who use them. But the fast growth of different IoT services and the changing needs of users have made it harder to manage limited edge resources efficiently. In MEC-enabled networks, intelligent task offloading and service caching are especially important for making sure that the system is always responsive and can run for a long time [3]. By connecting centralized cloud servers with distributed edge nodes, cloud-assisted MEC [4] makes the system even more flexible. This hierarchical framework allows latency-sensitive tasks to be processed at the edge, while the compute-intensive or cache-miss tasks to be offloaded to the Cloud. However,

effectively coordinating the Cloud and edge nodes under the dynamic traffic and resource conditions remains a challenge, motivating the need for adaptive, learning-based optimization framework [5].

Efficient task offloading in MEC determines the offloading location, such as a task should be computed on the device itself, offloaded to a nearby edge server, migrated to a nearby BS, or sent it to the cloud. Critical of tasks, computing power requirements, good network conditions, and disparate service requirements make task offloading challenging. The existing static and rule-based approaches are not suitable for these real-world situations [6]. Since edge nodes lack sufficient storage space to cache all the computing services, service caching becomes even more complicated. It becomes very challenging to determine which services to cache, when to update the cached services, when to replace the cached service to optimize the storage, and what should be done when cache miss occurs. If caching strategies aren't working well, services may go down often at the edge, the system may rely more on cloud offloading, and performance may suffer [7].

Furthermore, task offloading and service caching operate at different time scales. Task offloading decisions must be made quickly to respond to real-time task arrivals and network changes. In contrast, service caching and replacement are performed less frequently i.e. on longer timescales because of storage limits and the costs involved in deploying and updating services at the edge. Coordinating these decisions across various timescales while considering fluctuating traffic intensities continues to be a primary challenge in cloud-assisted MEC systems [8].

System Model

We consider a cloud-assisted mobile edge computing (CMEC) architecture composed of three logical layers, namely the IoT device layer, the fog (MEC) layer, and the remote cloud layer, as illustrated in Fig. 1. A set of heterogeneous IoT devices continuously generates computation-intensive service requests that must be processed with low latency. These requests are transmitted to nearby MEC-enabled base stations (BSs), which act as fog nodes and provide localized computation and caching capabilities. When local resources are insufficient or the requested services are unavailable at the edge, tasks are migrated to the cloud server, which offers virtually unlimited computing capacity at the cost of higher latency.

Let $\mathcal{M} = \{1,2, \dots, M\}$ denote the set of MEC-enabled BSs and $\mathcal{U} = \{1,2, \dots, U\}$ denote the set of IoT devices. Each BS $m \in \mathcal{M}$ is equipped with limited CPU capacity C_m^{cpu} and cache storage capacity C_m^{cache}. Furthermore, let $\mathcal{S} = \{1,2, \dots, S\}$ represent the set of candidate services or applications that may be required to execute user tasks. Because storing all services at every fog node is infeasible, only a subset of services can be cached locally.

Time is discretized into decision epochs $t \in \mathcal{T}$. At each epoch, devices generate tasks according to a stochastic arrival process. Each arriving task i is characterized by a tuple $\langle s_i, d_i, c_i, \tau_i \rangle$, where s_i denotes the requested service type, d_i is input data size, c_i represents required CPU cycles, and τ_i is latency tolerance. Tasks must be executed either locally, at neighboring fog nodes, or at the cloud. To enable decentralized intelligence and scalability, each BS is modeled as an autonomous learning agent that observes its local environment and independently determines offloading and caching decisions without requiring global network knowledge.

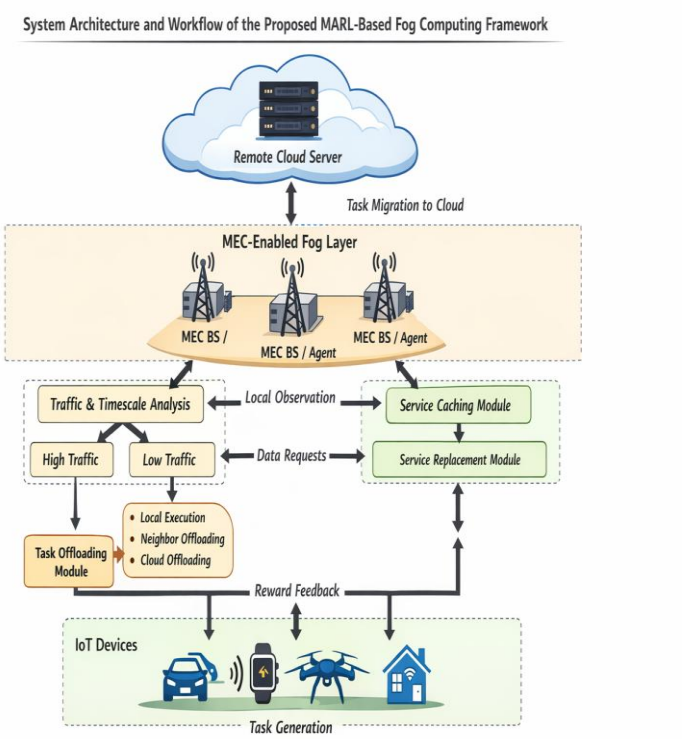

Fig. 1. System architecture and operational workflow of the proposed MARL-based fog computing framework.

Communication Model

Communication between devices, fog nodes, and the cloud occurs through wireless and backhaul links. Let $B_{u,m}$ denote the uplink bandwidth between device $u$and BS m, and $B_{m,n}$ represent the backhaul bandwidth between neighboring BSs. The achievable transmission rate is given by Shannon's capacity expression [9]

$$R_{u,m} = B_{u,m}\log_2(1 + \text{SNR}_{u,m}) \quad (1)$$

Accordingly, the transmission delay for task i offloaded from device u to BS m is defined in Eq. 2:

$$T_{u,m}^{tx}(i) = \frac{d_i}{R_{u,m}} \quad (2)$$

Similarly, offloading between fog nodes or toward the cloud incurs additional backhaul delays. This model captures the inherent trade-off between local execution and remote migration.

Computational Model

Each fog node processes tasks using its local CPU resources. Let f_m denote the available CPU frequency (cycles/s) of BS m. The execution time for task i processed at BS m is defined in Eq. 3.

$$T_m^{comp}(i) = \frac{c_i}{f_m} \quad (3)$$

If the service s_i required by the task is already cached at the BS, execution proceeds immediately. Otherwise, the service must be fetched from a neighbor or the cloud, incurring additional delay. Let T_m^{miss} denotes the service retrieval delay. The total execution latency becomes as follows:

$$L_m(i) = T_{u,m}^{tx}(i) + T_m^{comp}(i) + T_m^{miss} \quad (4)$$

Energy consumption at the fog node is modeled as

$$E_m(i) = \kappa f_m^2 c_i \quad (5)$$

where κ is the effective switched capacitance coefficient. This quadratic relationship captures the energy–frequency trade-off of dynamic voltage scaling.

For cloud execution, the computation delay is typically negligible compared to transmission delay; hence the latency is dominated by long-haul communication.

Service Caching Model

Each fog node maintains a cache storing frequently requested services to reduce service fetching latency. Let the binary variable $y_{m,s}^t \in \{0,1\}$ indicate whether service s is cached at BS m during time slot t. Because storage is limited, the following constraint must hold:

$$\sum_{s \in \mathcal{S}} y_{m,s}^t \cdot size_s \leq C_m^{cache} \quad (6)$$

Service demand follows time-varying popularity distributions. Let $p_{m,s}^t$ denote the probability that service s is requested at node m. Caching popular services increases the hit ratio and reduces both latency and backhaul usage. Moreover, when cache overflow occurs, services must be replaced. Replacement incurs migration overhead $R_{m,s}^t$, representing the cost of transferring service images between nodes. Hence, frequent updates should be avoided unless demand shifts significantly.

Task Offloading Model

For each task i, the offloading decision is represented by binary variable as shown in Eq. 7:

$$x_{i,m}^t = \begin{cases} 1, & \text{if task } i \text{ is processed at node } m, \\ 0, & \text{otherwise.} \end{cases} \quad (7)$$

Each task must be assigned to exactly one execution location as depicted in Eq. 8:

$$\sum_{m \in \mathcal{M} \cup \{\text{cloud}\}} x_{i,m}^t = 1 \quad (8)$$

The total computational load at any BS must not exceed its CPU capacity:

$$\sum_i x_{i,m}^t \, c_i \leq C_m^{cpu} \quad (9)$$

This constraint ensures queue stability and prevents congestion.

MARL-Based Decision Model

Due to the stochastic and time-varying nature of traffic, centralized optimization is impractical. Therefore, each BS is modeled as an agent within a decentralized partially observable Markov decision process (Dec-POMDP). At each time step, agent m observes local state:

$$s_m^t = \{q_m^t, u_m^t, y_m^t, b_m^t, d_m^t\} \quad (10)$$

where q_m^t denotes queue backlog, u_m^t CPU utilization, y_m^t cache status, b_m^t bandwidth, and d_m^t service demand. The agent selects action $a_m^t = (o_m^t, c_m^t)$, corresponding to tasks offloading and caching decisions.

The reward function is defined as [10]

$$r_m^t = -(\alpha L_m^t + \beta E_m^t + \gamma M_m^t + \delta R_m^t) \quad (11)$$

so that minimizing latency, energy, and cache misses maximizes cumulative reward. Each agent learns policy $\pi_m(a \mid s)$ through interaction with the environment.

Problem Formulation

The objective of the proposed system is to jointly determine offloading and caching decisions that minimize long-term operational cost across all fog nodes. Formally, the optimization problem is:

$$\min_{\mathbf{x},\mathbf{y}} \ \mathbb{E}\left[\sum_{t=0}^{T} \sum_{m\in\mathcal{M}} (\alpha L_m^t + \beta E_m^t + \gamma M_m^t + \delta R_m^t)\right] \quad (12)$$

subject to

$$\sum_m x_{i,m}^t = 1 \quad (12a)$$

$$\sum_i x_{i,m}^t \, c_i \le C_m^{cpu} \quad (12b)$$

$$\sum_s y_{m,s}^t \, size_s \le C_m^{cache} \quad (12c)$$

$$x_{i,m}^t, y_{m,s}^t \in \{0,1\} \quad (12d)$$

$$\alpha + \beta + \gamma + \delta = 1 \quad (12e)$$

This formulation constitutes a large-scale mixed-integer stochastic optimization problem that is NP-hard. Consequently, we approximate the optimal solution using decentralized multi-agent reinforcement learning, enabling real-time adaptive decision-making with significantly reduced computational overhead.

Multi-Agent Reinforcement Learning Framework

The decentralized decision-making problem is modeled as a Decentralized Partially Observable Markov Decision Process (Dec-POMDP) represented by the tuple $\langle \mathcal{M}, \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$. Each BS is treated as an independent agent that interacts with the environment and learns a local policy without requiring global state knowledge.

State Space: At time t, agent m observes local state $s_m^t = \{q_m^t, u_m^t, b_m^t, y_m^t, d_m^t\}$, where q_m^t is queue length, u_m^t CPU utilization, b_m^t available bandwidth, y_m^t cache status, and d_m^t service demand distribution.

Action Space: Each agent selects action $a_m^t = (o_m^t, c_m^t)$, where o_m^t determines the offloading destination (local, neighbor, or cloud) and c_m^t controls caching or replacement operations.

Reward Design: The reward function is defined as $r_m^t = -(\alpha L_m^t + \beta E_m^t + \gamma M_m^t + \delta R_m^t)$, ensuring agents learn to minimize overall system cost.

Policy Learning: Each agent learns policy $\pi_m(a \mid s; \theta_m)$ by maximizing the expected discounted return $J(\theta_m) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \, r_m^t\right]$. Neural networks approximate the policy, and parameters are updated using stochastic gradient descent over replay experiences.

Multi-Agent Task Offloading Strategy

During high-traffic periods, agents prioritize task scheduling. For each arriving task, the agent evaluates feasible destinations and selects the action that maximizes expected reward. Neighboring BSs are preferred when local resources are saturated, while the cloud serves as a fallback. This decentralized

coordination naturally balances load across the network and reduces congestion without centralized control.

Dynamic Service Caching via Knapsack Optimization

During low-traffic intervals, each agent optimizes its cache contents. Let $p_{m,s}$ denote estimated popularity of service s. The caching decision is formulated as follows:

$$\max_{y_{m,s}} \sum_s p_{m,s}\, y_{m,s} \text{s.t. capacity constraints} \quad (13)$$

This binary knapsack formulation ensures high-demand services are prioritized while respecting storage limits. The resulting cache configuration guides subsequent offloading decisions.

Reinforcement Learning-Based Service Replacement

When cache overflow occurs, a lightweight Q-learning mechanism selects which service to evict. State-action values are updated using Eq. 14 [11].

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)] \quad (14)$$

This adaptive policy learns long-term service utility and improves hit rates compared to heuristic approaches.

Algorithm 1 presents the pseudocode for the proposed strategy. The proposed framework operates as an iterative learning and optimization loop executed independently at each fog node. The proposed framework scales linearly with the number of BSs. Per-node complexity is $O(|\ S\ ||\ A\ |\ + S\log\ S)$ where the first term corresponds to MARL updates and the second to caching optimization.

Algorithm 1: Decentralized MARL-based Joint Task Offloading and Service Caching

Input: Set of fog nodes $\mathcal{M}$, Task arrivals from IoT devices, CPU capacities and cache capacities of nodes, Service popularity statistics, Learning parameters (learning rate, discount factor, episodes)

Output: Optimal offloading policy for each fog node, Optimal service caching configuration, Minimized latency, energy, and cache-miss cost

Start

1. Initialize Q-networks / policies for all fog agents
2. Initialize cache contents for each fog node
3. ***for*** each episode do
4. ***for*** each time step t do
5. ***for*** each fog node $m \in M$ (in parallel) do
6. Observe local state: queue length, CPU usage, bandwidth, cache status, service demand;
7. Select action using policy: $a_m \leftarrow$ (offloading decision, caching decision);
8. Execute action: process locally OR offload to neighbor/cloud;
9. Measure system metrics: latency, energy, cache misses;
10. Compute reward based on cost;
11. Update Q-value / policy using reinforcement learning;
12. End ***for***
13. ***if*** traffic is low then

```
        Solve knapsack optimization for cache placement
        Update cached services
    End if
  End for
End for
Return learned policies and cache configuration
End
```

Key Contributions

The main contributions of this work can be summarized as follows:

- Proposed a cloud-assisted MEC architecture integrating IoT devices, fog nodes, and cloud servers to support latency-sensitive applications.
- Developed a decentralized multi-agent reinforcement learning (MARL) framework for intelligent and scalable task offloading decisions.
- Formulated the joint task offloading and service caching problem as a Dec-POMDP, enabling distributed decision-making under dynamic network conditions.
- Designed a hybrid optimization strategy combining MARL-based task scheduling and knapsack-based service caching to efficiently utilize limited edge resources.
- Introduced an adaptive reinforcement learning-based service replacement policy to improve cache hit ratio and reduce service retrieval latency.

Conclusions and Future Works

- A decentralized MARL-based framework was proposed to jointly optimize task offloading and service caching in cloud-assisted MEC environments.
- The system architecture integrates IoT devices, fog nodes, and cloud infrastructure to support latency-sensitive applications with efficient distributed resource utilization.
- Modeling each fog node as an autonomous reinforcement learning agent enables scalable and adaptive decision-making without centralized coordination.
- The proposed strategy dynamically selects execution locations (local, neighbor, or cloud) to balance workload and reduce network congestion.
- A knapsack-based caching optimization combined with reinforcement learning-based service replacement improves cache efficiency under limited storage constraints.
- The framework is designed to minimize end-to-end latency, energy consumption, and cache-miss costs while improving service availability at edge nodes.
- Future work will implement the proposed framework in iFogSim over CloudSim to perform comprehensive experimental evaluation.
- The simulation study will analyze performance metrics such as latency, energy usage, cache hit ratio, scalability, and SLA violations, and compare the framework with baseline MEC scheduling and caching methods.

References

1. Pallewatta, K., Jayasinghe, D., and Weerasinghe, H., "Reliability-aware placement of microservices in fog computing environments using improved PSO and NSGA-II," *Journal of Network and Computer Applications*, vol. 237, pp. 103–117, 2024.
2. Chen, Y., Li, T., and Xu, D., "Personalized Federated Deep Reinforcement Learning for Computation Offloading and Resource Allocation in Smart Communities," *IEEE Transactions on Mobile Computing*, vol. 24, no. 3, pp. 1045–1060, 2024.
3. He, J., Wu, C., and Zhang, Q., "Joint Computation Offloading and Resource Allocation in Multi-Vendor Mobile-Edge Cloud Systems: A Game-Theoretic Approach," *IEEE Internet of Things Journal*, vol. 12, pp. 4462–4478, 2025.
4. Xie, J. and Cui, L., "Reward-Aware Proximal Policy Optimization for Dynamic Task Offloading in Mobile Edge Computing," *IEEE Transactions on Network and Service Management*, vol. 19, pp. 3152–3168, 2025.
5. Zhou, Y., Li, F., and Lin, Z., "QoE-aware Multi-user Task Offloading and Resource Allocation in MEC Systems using Binary Particle Swarm Optimization," *Computer Networks*, vol. 243, pp. 109680, 2025.
6. Naik, P., Das, A., and Ghosh, R., "Energy-efficient Workflow Scheduling in Edge Computing using Quantum-Inspired PSO," *Future Generation Computer Systems*, vol. 152, pp. 321–335, 2025.
7. Dai, S., Li, P., and Zheng, J., "Dynamic Service Caching and Task Offloading in Fog-Cloud Environments for Enterprise Systems," *IEEE Transactions on Cloud Computing*, vol. 13, pp. 1156–1170, 2023.
8. Shang, W., Li, Y., and Luo, M., "Deep Reinforcement Learning-based Joint Optimization for Computation Offloading and Service Caching in Edge-Cloud Networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 36, pp. 987–1002, 2024.
9. Wu, Q., Zhang, M., and Liu, Y., "Secure Task Offloading and Caching with Cooperative Jamming in MEC Networks," *IEEE Transactions on Mobile Computing*, vol. 24, pp. 1573–1587, 2024.
10. Cheng, L., Yang, T., and Zhang, K., "CO-MATCH: Joint Task Offloading and Service Caching for Vehicular Edge Computing," *IEEE Transactions on Intelligent Transportation Systems*, vol. 26, pp. 874–889, 2024.
11. Shang, C., Huang, Y., Sun, Y., & Guizani, M. (2024). Joint Computation Offloading and Service Caching in Mobile Edge-Cloud Computing via Deep Reinforcement Learning. IEEE Internet of Things Journal. 10.1109/JIOT.2024.3452117