Appl. Comput. Math., Vol.25, No.2, Special Issue, 2026, pp.205-226
DOI: 10.30546/1683-6154.25.2.2026.205

ADAPTIVE TASK OFFLOADING AND CACHING VIA REINFORCEMENT LEARNING IN MULTI-TIER MEDICAL CYBER-PHYSICAL SYSTEMS

S. QIAO[1], Q. XIE[2], X. HE[3,*], M. WANG[4,*], Q. DU[2], A. ALMUSHARRAF[5], M. SHABAZ[6]

Abstract. In multi-tier medical cyber-physical systems (MCPS), heterogeneous devices such as wearable and implantable medical equipment continuously generate a large volume of computational tasks. Due to limited local computing resources, these tasks need to be offloaded to edge servers (ESs). However, the efficiency of medical task execution depends on whether the required AI services are cached at the ESs, while caching strategies are simultaneously influenced by the dynamic arrival of tasks. This interdependence makes the joint optimization of task offloading and service caching highly challenging under random task arrivals. To address this challenge, we propose a Deep Reinforcement Learning-based Task Offloading and Service Caching optimization algorithm (DRL-TOSC) that jointly optimizes medical task offloading and AI service caching in MCPS. By integrating the Temporal Difference (TD) method, the medical agent can gradually learn the optimal policy through online interaction with dynamic environments, ensuring fast convergence and robust decision-making. Furthermore, we design a Valid Action Reallocation Mechanism (VARM) that improves decision efficiency by eliminating invalid actions and renormalizing the probability distribution of valid ones. Experimental results demonstrate that DRL-TOSC reduces the average system cost by 42.3% and improves the service cache hit rate by 20.7% compared to existing benchmark algorithms.

Keywords: Medical Cyber-Physical Systems, Deep Reinforcement Learning, AI Service Cache, Real-Time Data Processing, Intelligent Decision Support.

AMS Subject Classification: 68M14, 68T05, 68W27.

1. Introduction

Medical cyber-physical systems (MCPS) tightly integrate sensing, computing, and control across wearable or implantable devices, bedside monitors, local servers in clinical settings, and cloud services [7, 14]. Common applications such as continuous heart rhythm and blood oxygen monitoring, image-based diagnosis using ultrasound or CT scans, or automated treatment through insulin pumps and cardiac devices-require rapid data processing and reliable connections to support real-time alerts and clinical decision-making [21]. Relying solely on cloud servers cannot meet the stringent requirements of speed, reliability, and privacy protection, while relying solely on medical devices (MDs) is constrained by limited battery life, storage capacity,

[1]School of Software, Tiangong University, Tianjin, China, e-mail: siboqiao@tiangong.edu.cn
[2]School of Control Sciences and Engineering, Tiangong University, Tianjin, China
e-mail: 2331091181@tiangong.edu.cn, duqifei@tiangong.edu.cn
[3]Qingdao Institute of Software, College of Computer Science and Technology, China University of Petroleum (East China), Shandong, China
[4]School of Life Sciences, Tiangong University, Tianjin, China
[5]Department of Management, College of Business Administration, Princess Nourah bint Abdulrahman University, P.O. Box 84428, Riyadh 11671, Saudi Arabia, e-mail: aalmusharraf@pnu.edu.sa
[6]Marwadi University Research Center, Marwadi University, Rajkot, Gujarat, India
e-mail: bhatsab4@gmail.com
*Corresponding author e-mail: b23070003@s.upc.edu.cn, wangmin@tiangong.edu.cn
Manuscript received 15 September 2025.

and computing power [11, 19, 30]. Therefore, a multi-tier end-edge-cloud architecture is essential: MDs are responsible for real-time sensing, nearby medical edge servers (ESs) perform rapid analysis and data integration, and the cloud performs deep analysis, long-term learning, and model updates [35].

However, in MCPS, the workload of MDs fluctuates dynamically depending on patients physiological states; for example, a sudden increase in heart rate or an abnormal blood glucose level may trigger additional data collection and processing demands [15]. Meanwhile, the quality of wireless links and backhaul capacity varies considerably across time and location [13], imposing stringent requirements on real-time responsiveness and system robustness. As a result, when making task offloading decisions, MDs must minimize response latency and energy consumption while operating under limited bandwidth constraints [6, 25]. Moreover, MCPS tasks are highly heterogeneous, with requirements that differ significantly across clinical services and network conditionsfor instance, image-based diagnosis, vital sign monitoring, and automated drug delivery [8]. These tasks demand diverse AI services, and efficient execution is achievable only when the required AI services are cached at the ESs [27]. Due to limited storage capacity, ESs cannot cache all AI services simultaneously, which necessitates intelligent caching strategies to accommodate dynamic and heterogeneous task arrivals [2]. As a result, medical task offloading and AI service caching are tightly coupled, and their joint optimization emerges as a fundamental challenge in MCPS.

Existing studies have proposed various joint optimization methods for medical task offloading and edge AI service caching in MCPS. For example, some studies design heuristic-based coordination schemes [20], while others employ optimization frameworks or machine learning models to balance offloading efficiency and caching effectiveness, thereby improving system performance and resource utilization. Traditional heuristic and optimization methods, however, rely heavily on static modeling and prior knowledge, making it difficult to adapt and converge in dynamic and complex environments. In contrast, deep reinforcement learning (DRL) can interact directly with the environment [28], continuously learning and refining policies to support medical task offloading decisions and real-time task analysis. Recent DRL-based studies [9, 26] have explored resource allocation, adaptive offloading, and service caching in edge-assisted healthcare, demonstrating improved adaptability and performance compared to static optimization approaches. Achieving efficient large-scale task processing in dynamic MCPS still faces two key challenges: (i) in MCPS, medical task offloading and edge AI service caching are strongly coupled, and their interaction directly determines system latency, energy consumption, and communication cost. Moreover, link resources are inherently limited, and improper allocation of link capacity makes it difficult to ensure reliable transmission of medical tasks. How to fully utilize constrained resources while jointly optimizing offloading and caching remains a critical problem [5]; and (ii) inefficient exploration during DRL training in MCPS, where the medical agent frequently executes invalid actions, such as offloading medical tasks to servers lacking the required AI services, which reduces learning efficiency, while overly strong penalty mechanisms may lead to conservative policies and suboptimal convergence [12].

To address these challenges, we propose a medical task offloading and edge AI service caching optimization framework for MCPS. From the perspective of joint optimization under taskcaching coupling and limited link capacity, we model medical task offloading and AI service caching as a Markov Decision Process (MDP) to minimize latency and energy consumption while efficiently utilizing limited link resources. In the aspect of policy learning and exploration efficiency, we design a Deep Reinforcement Learning-based Task Offloading and Service Caching algorithm (DRL-TOSC), where the Temporal Difference (TD) method enables the medical agent to gradually approach the optimal policy in dynamic environments, and a Valid Action Reallocation Mechanism (VARM) eliminates invalid actions and redistributes probabilities over valid ones, thereby enhancing exploration efficiency and decision quality. The key contributions are outlined as follows.

1) We incorporate link capacity constraints into the system model for MCPS, enabling real-time load-aware medical task offloading and jointly optimizing task latency and energy consumption within the MDP framework.

2) We propose a DRL-TOSC algorithm that integrates the Temporal Difference (TD) for fast convergence and adaptability, and incorporates VARM to eliminate invalid offloading actions and renormalize valid ones, improving decision efficiency and reducing policy complexity.

3) We conduct extensive evaluations across multiple scenarios against PPO, DDPG, OOA, RES, EES, and LES. The results demonstrate that DRL-TOSC consistently outperforms baselines, reducing average system cost by 42.3% and increasing the service cache hit rate by 20.7%.

The rest of the work is organized as follows. Section 2 reviews related work. Section 3 presents the system model and problem formulation. Section 4 details the algorithm design. Section 5 discusses the experimental evaluation results and Section 6 offers the conclusion and future research.

2. Related work

With the continuous advancement of research on medical task offloading and edge AI service caching, numerous significant contributions have emerged in MCPS [1, 16, 23, 29]. In [1], the authors addressed load imbalance in IoT-enabled smart healthcare by proposing a dynamic load-balancing scheme based on fog computing, which adaptively distributes medical tasks across edge resources, thereby improving latency, throughput, and network performance. In [23], the author proposed an agentic AI framework for dynamic health management and real-time genomic analytics, outlining autonomous workflows that integrate multi-modal medical data to support continuous monitoring and clinical decision-making. In [16], the authors studied energy-efficient medical task offloading and resource allocation in augmented realityassisted healthcare applications within edge computing, employing a DRL-based MADDPG algorithm to reduce terminal energy consumption. In [29], the authors investigated optimal joint scheduling and cloud offloading for mobile healthcare applications, introducing wireless-aware joint scheduling and computation offloading with mathematical modeling and linear optimization to capture dependency constraints. These studies provide various resource allocation mechanisms; however, in practice, medical task execution depends on specific AI service components such as libraries, executables, and clinical input data [14], which requires ESs to cache both services and their dependencies.

The overhead of medical data transmission is inherently coupled with task offloading decisions, directly affecting performance and resource allocation in MCPS. Several studies have addressed this issue [4,34,37,38]. In [37], the authors examined multi-task inference in vehicular health monitoring and introduced a joint model deployment and task offloading scheme with shared sensing to minimize response time. In [4], the authors proposed CachOf, which enhances healthcare application efficiency in MEC by combining sub-task priority computation, dynamic caching, and DRL-driven offloading. In [38], the authors developed a joint optimization model for service caching placement, computation offloading, and system resource allocation, aiming to conserve device resources through alternating minimization. In [34], the authors investigated service caching and task offloading in UAV-assisted MCPS, introducing an OOA that dynamically updates cache strategies, optimizes UAV trajectories, and refines task offloading to reduce service latency under energy constraints. Nevertheless, most existing methods incompletely model link conditions and fail to jointly consider latency and energy consumption, making it difficult to balance performance and efficiency with a weighting factor. In data-intensive healthcare applications, link bandwidth limitations directly affect the success rate and efficiency of medical task offloading. Therefore, considering link capacity constraints while jointly optimizing latency and energy consumption is critical for improving MCPS performance.

More recently, studies have investigated the joint optimization of medical task offloading and AI service caching in MCPS [3, 33, 36]. In [3], the authors explored collaborative service caching and computational offloading in IoT-based healthcare, employing a hierarchical DRL approach to optimize task execution time and energy efficiency. In [36], the authors studied cache allocation in multi-tenant healthcare edge systems, leveraging an online model-based RL method to allocate cache under encrypted and partially non-cacheable content. In [33], the authors applied a DRL-based framework to collaborative healthcare applications, enhancing resource efficiency and load balancing through joint caching and resource allocation. Some DRL-based approaches attempt to suppress invalid actions by increasing penalties in the reward function, which helps avoid invalid offloading decisions but often leads on training instability or local optima. By contrast, action masking mechanisms dynamically exclude invalid actions, improving learning efficiency and decision quality. Hence, when comparing existing methods, it is essential to evaluate their ability to constrain policies and maintain exploration efficiency under a unified action space representation.

In summary, medical task offloading and AI service caching are highly coupled in MCPS. In dynamic network environments with random medical task arrivals and limited link capacity, jointly optimizing these strategies to improve resource utilization and reduce system overhead remains a key challenge that traditional heuristic methods cannot effectively address. To this end, we propose DRL-TOSC, which integrates the TD method to enhance convergence and adaptability in dynamic MCPS, and introduce VARM to improve decision efficiency by eliminating invalid actions and renormalizing the probability distribution.

TABLE 1. Description of the main symbols.

Symbol	Description
I	Set of MDs
Q	Set of AI service types in edge caching
$x_i(t)$	Size of the original input data
$\rho_i(t)$	CPU computation required for task execution
$q_i(t)$	Service cache type requested by the task
$y_i(t)$	Size of the output data after task processing
$K(t)$	Set of caching decisions at ES
$k_q(t)$	Caching decision for service q at time slot t
s_q	Storage space required for each service $q \in Q$
$Z(t)$	Set of offloading decisions for MD
P_{st}	Standby power of the MD
P_{rx}	Power consumption for result transmission
$T(t)/E(t)$	Total task delay/energy consumption
R_c	Rate of data transmission to the cloud
$R_{t'}$	The immediate reward at time step t'
d_i	Distance from MD i to the server
V_t^*	Target value in TD method
$U_i(t)$	Total cost of task execution
$\varepsilon_t/\varepsilon_e$	Preference parameters for latency/energy consumption

3. System model and problem formulation

This section introduces the MCPS model, edge AI service caching model, MCPS task computation model, and the problem definition and formulation. The key notations are summarized in Table 1.

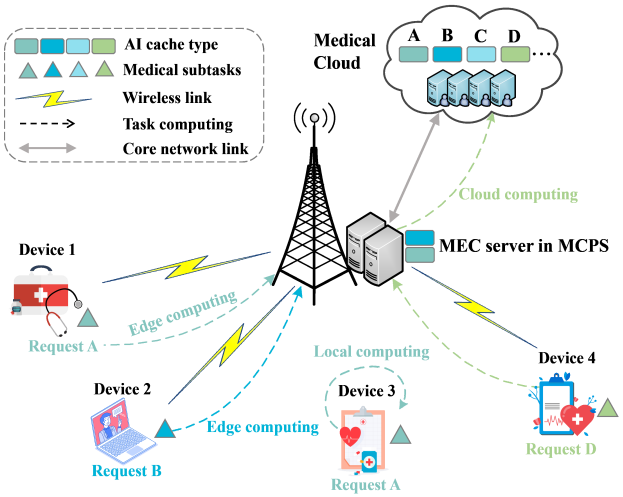


FIGURE 1. MCPS model.

3.1. **MCPS Model.** Fig.1 depicts a hierarchical collaborative computing architecture consisting of the cloud, a MEC cluster of ESs, and distributed MDs. The MEC servers integrate multimodal service caching functionality and enable dynamic, coordinated service resource scheduling through the computing cluster. The MEC network is composed of multiple edge nodes, supporting dynamic task offloading and service distribution, which select the optimal node based on terminal requests and cache status. When the required AI service is available in the cache, the medical task is executed at the edge. If the computational demand exceeds the capacity of the MD, the MCPS task is offloaded to the ES. Complex medical tasks beyond edge capabilities are further forwarded to the cloud for execution [24]. Our model assumes structural heterogeneity among MCPS tasks associated with the same cached AI service. For example, in medical imaging analysis [22], variations in resolution, modality, and preprocessing requirements lead to diverse computational demands.

For clarity, we define the set of MDs as $I = \{1, 2, \ldots, i\}$ and the set of AI service types cached at the edge as $Q = \{1, 2, \ldots, q\}$. The time slot t represents the smallest unit of time for communication and resource allocation. At each time slot t, the MCPS task generated by MD i is represented as a four-tuple $\mu_i(t) = \{x_i(t),\ \rho_i(t),\ q_i(t),\ y_i(t)\}$, where $x_i(t)$ denotes the size of the raw input data, $\rho_i(t)$ represents the CPU cycles required for task execution, $q_i(t)$ indicates the type of AI service requested by the MCPS task, and $y_i(t)$ refers to the size of the output data after processing. Following [3, 33], we assume that the cloud can cache all types of AI services, whereas MEC servers, due to storage limitations, can only cache a limited subset.

3.2. **Edge AI Caching Model.** In MCPS, ESs can cache frequently used AI services to enable rapid response during peak periods [33]. Initially, all medical AI services are stored in the cloud, and ESs selectively download and cache specific services locally to reduce frequent cloud access, thereby improving task processing speed and response efficiency. We define the caching decision set of the ES at time slot t as $K(t) = \{k_1(t), k_2(t), \ldots, k_Q(t)\}$, where the binary caching variable $k_q(t) \in \{0, 1\}$ represents the caching status of service q. Specifically, $k_q(t) = 0$ indicates that service q is not cached at the server during time slot t, while $k_q(t) = 1$ indicates that the service is cached. The storage space occupied by cached services must not exceed the available storage capacity of the ES [32]. In practice, this model can be combined with DRL to optimize caching strategies in real time, enabling adaptive updates and on-demand service deployment. This constraint is formulated as follows.

$$\sum_{q=1}^{Q} k_q(t) s_q \leq C_s, q \in Q, \tag{1}$$

where s_q represents the storage space required for each service $q \in Q$, and C_s denotes the available storage capacity of the ES. ES and the cloud process download and upload tasks in parallel via a high-speed wired connection, ensuring that dynamic service caching does not introduce additional latency.

3.3. **Computation Model.** In MCPS, medical tasks can be executed in three computing modes: Local execution on the MD, offloading to the ES if the required AI service is cached, or offloading to the cloud for execution.

At time slot t, we define the offloading decision set of the MD as $Z(t) = \{\varphi_1(t), \ldots, \varphi_I(t)\}$, where $\varphi_i(t)$ is given by $\varphi_i(t) = \{\alpha_i^l(t), \beta_i^e(t), \gamma_i^c(t)\}$. $\alpha_i^l(t) = 1$ indicates local execution, $\beta_i^e(t) = 1$ represents offloading to ES, and $\gamma_i^c(t) = 1$ denotes offloading to the remote cloud. These three decision variables satisfy the constraint expressed as $\alpha_i^l(t) + \beta_i^e(t) + \gamma_i^c(t) = 1$. The following sections present the computation model in three parts.

3.3.1. *Local computing.* At time slot t, if a MCPS task is executed locally on the MD without offloading, the main concern is the execution latency. Since different MDs have varying computational capabilities, we denote the local computing capacity of MD i as f_i^l. The corresponding task execution latency and energy consumption are formulated as:

$$T_i^l(t) = \frac{\rho_i(t)}{f_i^l}, \tag{2}$$

$$E_i^l(t) = \eta (f_i^l)^2 \rho_i(t), \tag{3}$$

where η is the energy consumption coefficient.

3.3.2. *Edge computing.* At time slot t, the MD offloads medical task data to the ES, where the computation is performed and the results are returned. According to Shannon's formula, the uplink and downlink transmission rates between device i and the ES are given by:

$$R_i^u(t) = W \log_2 \left(1 + \frac{P_{\mathrm{MD}} h_i^u(t)}{\sigma^2}\right), \tag{4}$$

$$R_i^d(t) = W \log_2 \left(1 + \frac{P_{\mathrm{ES}} h_i^d(t)}{\sigma^2}\right), \tag{5}$$

where W denotes the channel bandwidth, P_{MD} and P_{ES} represent the transmission power of the MD and the ES, respectively, σ^2 is the additive white Gaussian noise power, and $h_i^u(t)$ and $h_i^d(t)$ are the uplink and downlink channel fading coefficients. We assume that the ES evenly distributes its computational resources, ensuring that each device receives sufficient processing capacity for efficient task execution. The total processing time and energy consumption of a medical task on the ES are expressed as:

$$T_i^e(t) = \frac{x_i(t)}{R_i^u(t)} + \frac{\rho_i(t)}{f_e / \sum_{i=1}^{I} \beta_i^e(t)} + \frac{y_i(t)}{R_i^d(t)}, \tag{6}$$

$$E_i^e(t) = P_{\mathrm{MD}} \frac{x_i(t)}{R_i^u(t)} + P_{\mathrm{st}} \frac{\rho_i(t)}{f_e / \sum_{i=1}^{I} \beta_i^e(t)} + P_{\mathrm{rx}} \frac{y_i(t)}{R_i^d(t)}, \tag{7}$$

where f_e represents the computing capacity of the ES, P_{st} denotes the standby power of the device, and P_{rx} is the power consumption for result transmission.

3.3.3. *Cloud computing.* At time slot t, when a MD offloads a medical task to the cloud, the request data is first transmitted to the ES via the wireless network and then forwarded to the remote cloud through the core network [1]. The overall processing latency and energy consumption for executing the MCPS task in the remote cloud can be expressed as follows.

$$T_i^c(t) = \frac{x_i(t)}{R_i^u(t)} + \frac{x_i(t) + y_i(t)}{R_c} + \frac{y_i(t)}{R_i^d(t)}, \tag{8}$$

$$E_i^c(t) = P_{\mathrm{MD}} \frac{x_i(t)}{R_i^u(t)} + P_{\mathrm{st}} \frac{x_i(t) + y_i(t)}{R_c} + P_{\mathrm{rx}} \frac{y_i(t)}{R_i^d(t)}, \tag{9}$$

where R_c denotes the data transmission rate to the cloud.

In summary, considering the three computing modes, the overall medical task delay and energy consumption can be expressed as:

$$T(t) = \alpha_i^l(t) T_i^l(t) + \beta_i^e(t) T_i^e(t) + \gamma_i^c(t) T_i^c(t), \tag{10}$$

$$E(t) = \alpha_i^l(t) E_i^l(t) + \beta_i^e(t) E_i^e(t) + \gamma_i^c(t) E_i^c(t). \tag{11}$$

We define ε_t and ε_e as the preference parameters for MCPS delay and energy consumption, respectively, satisfying $\varepsilon_t + \varepsilon_e = 1$. The total cost of medical task i at time t is defined as a weighted sum of latency and energy consumption: $U_i(t) = \varepsilon_t T(t) + \varepsilon_e E(t)$, where ε_t and ε_e represent the preference weights for latency and energy, respectively. The total cost for executing all tasks is given by:

$$U(t) = \sum_{i=1}^{I} U_i(t) = \sum_{i=1}^{I} \left[\varepsilon_t T(t) + \varepsilon_e E(t) \right]. \tag{12}$$

3.3.4. *Link Load Capacity Constraints.* In MCPS, link resources often constitute a critical bottleneck [23]. To prevent medical data transmission from exceeding link bandwidth limits, we introduce a link load capacity constraint into the model. This ensures that, at any time slot t, the total transmission volume does not exceed the maximum link capacity, thereby avoiding congestion and improving resource utilization. The link must simultaneously carry uplink offloading data from medical devices, intermediate task data, and feedback of offloading results. We separately model the bandwidth constraints of wireless and wired links as follows.

$$\begin{aligned} \sum_{i \in I} \left(\beta_i^e(t) R_i^u(t) + \gamma_i^c(t) \frac{x_i(t) + y_i(t)}{\Delta T} \right) &\leq C_L^{\mathrm{wireless}}, \\ \sum_{i \in I} \left(\left(\beta_i^e(t) + \gamma_i^c(t) \right) R_i^d(t) \right) &\leq C_L^{\mathrm{wired}}, \end{aligned} \tag{13}$$

where ΔT denotes the time slot duration, C_L^{wireless} is the wireless link capacity, and C_L^{wired} represents the wired link capacity (in Mbps).

3.4. **Problem Definition and Formulation.** To jointly optimize medical task offloading and edge AI service caching, the objective is to minimize the overall execution delay and energy consumption of medical tasks subject to ES storage constraints and link capacity constraints. The joint optimization problem is formulated as follows:

$$
\begin{aligned}
\mathbf{P}: \quad & \min_{\{K(t),Z(t)\}} \sum_{i=1}^{I} U_i(t), \\
\text{s.t.} \quad & \\
\mathbf{C}_1: \quad & k_q(t) \in \{0,1\}, \forall q \in Q, \\
\mathbf{C}_2: \quad & \sum_{q=1}^{Q} k_q(t) s_q \leq C_s, \forall q \in Q, \\
\mathbf{C}_3: \quad & \beta_i^e(t) \leq k_q(t), \forall q \in Q, i \in I, \\
\mathbf{C}_4: \quad & \alpha_i^l(t) + \beta_i^e(t) + \gamma_i^c(t) = 1, \forall i \in I, \alpha_i^l(t), \beta_i^e(t), \gamma_i^c(t) \in \{0,1\}, \\
\mathbf{C}_5: \quad & \sum_{i \in I} \left(\beta_i^e(t) R_i^u(t) + \gamma_i^c(t) \frac{x_i(t) + y_i(t)}{\Delta T} \right) \leq C_L^{\text{wireless}}, \\
& \sum_{i \in I} \left(\left(\beta_i^e(t) + \gamma_i^c(t) \right) R_i^d(t) \right) \leq C_L^{\text{wired}}.
\end{aligned} \tag{14}
$$

Constraint $\mathbf{C}_1$ specifies the caching state of each AI service q (cached or not cached). Constraint $\mathbf{C}_2$ ensures that the storage allocated for cached AI services does not exceed the available storage capacity of the ES. Constraint $\mathbf{C}_3$ defines the dependency between medical task offloading and the availability of the required AI services on the ES. Specifically, $k_q(t) \in \{0,1\}$ indicates whether the AI service type q requested by a MD is cached at time slot t. This constraint guarantees that medical tasks can only be offloaded to an ES if the required AI service is already cached, thereby maintaining consistency between service availability and offloading decisions. Constraint $\mathbf{C}_4$ enforces that each MD i can only select one computation mode per time slot t: Local execution on the MD, offloading to an ES (if the required service is cached), or offloading to the cloud. Constraint $\mathbf{C}_5$ introduces the link load capacity constraint, ensuring that medical data transmission does not exceed the available network resources, thereby preventing congestion and improving link utilization efficiency.

The optimization problem $\mathbf{P}$ is subject to time-varying wireless channels and dynamic medical task arrivals. The transmission cost depends on the task data size and available channel bandwidth, while the computational cost is determined by the processing workload. Consequently, a trade-off must be established between local resource utilization and medical edge/cloud resource allocation. Traditional optimization typically require require repeated iterations and are often inefficient for handling rapidly changing MCPS task demands [3]. Therefore, we formulate problem $\mathbf{P}$ as a MDP and employ a DRL-based approach to derive efficient offloading and caching strategies.

4. Algorithm design

In MCPS, this section begins with an overview of the MDP. Next, the fundamental principles of DRL are introduced. Finally, the proposed algorithm is elaborated in detail.

4.1. **MDP construction.**

To address the joint optimization problem of medical task offloading and edge AI service caching in MCPS, we model it as an MDP with the objective of minimizing overall task execution delay and energy consumption. In this framework, the medical ES functions as the medical agent, collecting medical system state information and making offloading and caching decisions at each time slot based on reward feedback from the environment. The subsequent subsections define the medical agents state space, action space, and reward function in detail.

(1) State Space: At time slot t, the medical ES observes system parameters such as channel bandwidth, computational frequency, and storage capacity. The agent extracts features from the input environmental state $s_t \in S$ and makes decisions. The state space can be represented as:

$$s_t = \{x_i(t), \rho_i(t), q_i(t), y_i(t), h(t), C_L\}, \tag{15}$$

where $h(t) = \{h_1^u(t), \ldots, h_i^u(t), h_1^d(t), \ldots, h_i^d(t)\}$ denotes the channel state of device i at time slot t.

(2) Action Space: At time slot t, the medical agent optimizes the ES's caching strategy by caching popular services. The action space A includes both offloading and caching decisions, with the action $a_t \in A$ is expressed as:

$$a_t = \{K(t), \alpha_i^l(t), \beta_i^e(t), \gamma_i^c(t)\}, \tag{16}$$

where $K(t) = \{k_1(t), k_2(t), \ldots, k_Q(t)\}$ represents caching decisions for Q services.

(3) Reward Function: At time slot t, the objective is to minimize task delay and device energy consumption. Since DRL seeks to maximize the reward, the reward function is inversely related to the objective function. The reward function $R(t)$ is formulated as:

$$R(t) = -U(t) = -\sum_{i=1}^{I} \left[\varepsilon_t T(t) + \varepsilon_e E(t)\right]. \tag{17}$$

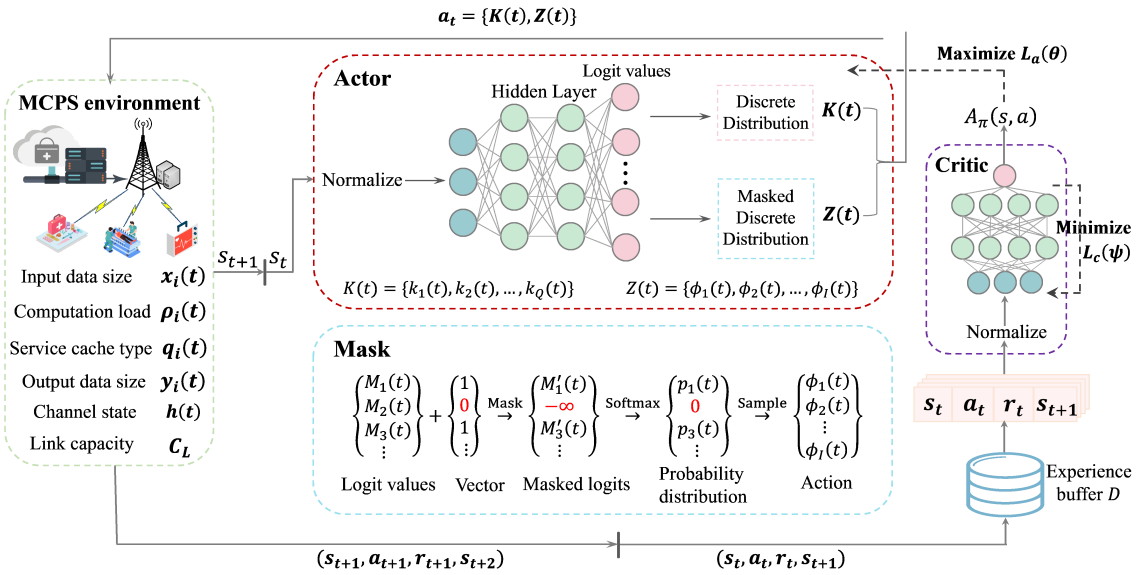


FIGURE 2. Framework of the DRL-TOSC algorithm in MCPS.

4.2. **Core DRL principles.** In the core theory of DRL, the state-value and action-value functions assess the expected returns of a state and state-action pair, respectively. The advantage function quantifies the relative benefit of a specific action, while the TD method integrates sampling and dynamic programming for efficient value estimation in MCPS. The following subsections provide a detailed discussion.

(1) State-Value Function: Under policy π, the expected cumulative return starting from state s is given by:

$$J_\pi(s) = \mathbb{E}_\pi \left[\sum_{t'=t}^{\infty} \gamma^{t'-t} R_{t'} \mid S_t = s\right], \tag{18}$$

where the discount factor $\gamma \in [0,1]$, t' represents a future time step starting from the current time step t, $R_{t'}$ is the immediate reward at time step t', and $\gamma^{t'-t}$ is used to weight future rewards, emphasizing their diminishing impact over time.

(2) Action-Value Function: Under policy π, the expected cumulative return after taking action a in state s is given by:

$$G_\pi(s,a) = \mathbb{E}_\pi \left[\sum_{t'=t}^{\infty} \gamma^{t'-t} R_{t'} \mid S_t = s, A_t = a \right], \tag{19}$$

where future rewards $R_{t'}$ are weighted by $\gamma^{t'-t}$ to reduce their influence on current decision-making, thereby accounting for the effect of temporal delay.

(3) Advantage Function: The advantage function $A_\pi(s,a)$ quantifies the relative benefit of taking action a compared to the expected return under policy π, guiding policy optimization.

$$A_\pi(s,a) = G_\pi(s,a) - J_\pi(s). \tag{20}$$

(4) Temporal Difference (TD) Residual: The TD residual quantifies the difference between the current value estimate and the prediction derived from immediate rewards and the value of the next state.

$$\Delta_t = R(t) + \gamma J(s_{t+1}) - J(s_t). \tag{21}$$

4.3. **Algorithm Introduction.** This section first outlines the theoretical foundations of the algorithm in MCPS, followed by a detailed discussion of its design and implementation.

4.3.1. *Theoretical Foundation.* Fig.2 illustrates the framework of the DRL-TOSC algorithm for MCPS, which consists of two main components: the Actor, responsible for decision-making, and the Critic, responsible for evaluation. The environment encompasses multiple factors relevant to medical task offloading and edge AI service caching, including input data size, computational workload, requested AI service type, output data size, and wireless channel state, all of which collectively influence the medical agents decision-making process. The medical agents action is defined as: $a_t = \{K(t), Z(t)\}$, where $K(t)$ denotes the caching decision and $Z(t)$ the offloading decision at time slot t. The Actor network adopts a fully connected architecture with two hidden layers of 256 neurons each. Based on the current environmental state, it generates policies and outputs the probability distribution of possible actions, which is further normalized by the Softmax function to select the final action. The Critic evaluates the value of the current policy and refines the Actors decision-making by leveraging the TD method to optimize the value function. An experience replay mechanism is employed to iteratively update the policy, maximizing the policy loss and minimizing the value function loss.

Due to the interdependence between medical task offloading and AI service caching, the action space expands rapidly as the number of feasible action combinations increases. To mitigate this, we adopt an action composition strategy by decomposing the decision into sub-actions. Specifically, at time slot t, the offloading decision for device i is denoted as $\varphi_i(t) \in \{0,1,2\}$, corresponding to local execution, offloading to the edge, and offloading to the cloud, respectively.

If the required service is not cached on the MEC server, offloading the task to the edge is rendered invalid. In reinforcement learning, a common approach to restrict invalid actions is to assign a large negative value to the reward function [34]. However, this method cannot entirely prevent invalid actions and struggles to obtain effective rewards during the exploration phase. To overcome this limitation, we propose a VARM, which eliminates invalid actions and adjusts the probability distribution of valid actions rather than simply applying penalties.

To distinguish between valid and invalid actions, we introduce a binary variable

$$\lambda_j(t) = \begin{cases} 0, & \text{invalid action.} \\ 1, & \text{valid action.} \end{cases}$$

This variable is determined based on the available service information from MDs and medical task requests. The masked logit value $M'_j(t)$ for action j is formulated as:

$$M'_j(t) = M_j(t) + (1 - \lambda_j(t)) \times N, \tag{22}$$

where N is a significantly large negative value, ensuring that the probability of invalid actions approaches zero after Softmax processing. This approach effectively eliminates invalid actions, ensuring that the MCPS performs random sampling exclusively within the valid action set, thereby significantly enhancing learning efficiency and overall performance.

(1) Actor Network: The probability ratio $p_t(\theta)$ is used to measure the difference between the updated and previous policy under the same state-action pair.

$$p_t(\theta) = \frac{\pi_{\text{after}}(a_t \mid s_t)}{\pi_{\text{before}}(a_t \mid s_t)}. \tag{23}$$

To prevent excessive policy updates, we adopt the clipping technique from the PPO-clip algorithm [32], which restricts the probability ratio $p_t(\theta)$ within the range $[1-\sigma, 1+\sigma]$ to avoid over-optimization. The clipping function is defined as:

$$\text{clip}(p_t(\theta), 1-\sigma, 1+\sigma) = \begin{cases} 1-\sigma, & p_t(\theta) \le 1-\sigma \\ 1+\sigma, & p_t(\theta) \ge 1+\sigma \\ p_t(\theta), & 1-\sigma \le p_t(\theta) \le 1+\sigma. \end{cases} \tag{24}$$

The objective function of the Actor Network is given by:

$$L_a(\theta) = \hat{\mathbb{E}}_t \left[\min \Big(p_t(\theta) A_\pi(s,a), \text{clip}(p_t(\theta), 1-\sigma, 1+\sigma) A_\pi(s,a) \Big) + \xi H \right], \tag{25}$$

where ξ is the entropy weight and H the policy entropy, which promotes exploration and helps avoid premature convergence to local optima.

(2) Critic Network: The objective function of the Critic Network is formulated as:

$$L_c(\psi) = \hat{\mathbb{E}}_t \left[(V_\psi(s_t) - V_t^*)^2 \right], \tag{26}$$

where $V_\psi(s_t)$ represents the state-value function, which evaluates the value of the current state s_t, while V_t^* is the target value derived from the TD method, used to update the state-value function. It is defined as

$$V_t^* = R(t) + \gamma J(s_{t+1}). \tag{27}$$

The TD method does not require additional hyperparameter tuning, thereby significantly reducing computational complexity. By leveraging the bootstrapping mechanism, it efficiently estimates state values, thereby improving the stability and convergence speed of the algorithm in MCPS. To ensure practical applicability, we also consider security implications of task offloading in MCPS: edge execution should occur within trusted environments, data in transit must be protected via end-to-end encryption, and both tasks and returned results should be authenticated and verified. These safeguards help preserve confidentiality and integrity while supporting stable, convergent learning in deployment.

Algorithm 1 Deep Reinforcement Learning-based Task Offloading and Service Caching Algorithm (DRL-TOSC)

Initialize: Set weights and biases for policy network $\pi_{after}(a_t|s_t)$ and value network $V_\psi(s_t)$; Initialize experience replay buffer D;
# Train using experience medical data from policy $\pi_{before}(a_t|s_t)$;
while iter < max_iter **do**
  for each time slot $t = 0$ to T **do**
    Record the current state s_t;
    Under the link capacity constraint, valid actions are renormalized, and $\pi_{before}(a_t|s_t)$ is executed to generate the joint decision;
    Perform the medical device action a_t;
    Compute the immediate reward $R(t)$ by (17);
    Record the next state s_{t+1};
    Store (s_t, a_t, r_t, s_{t+1}) in buffer D;
  end for
  # Optimize the neural networks;
  for epoch = 1 to U **do**
    Shuffle data in D into small batches to compute gradients and update networks;
    Compute the advantage function Δ_t by (21);
    Use the Adam optimizer to maximize $L_a(\theta)$ and update policy network parameters;
    Use the Adam optimizer to minimize $L_c(\psi)$ and update value network parameters;
  end for
  Update $\theta_{before} \leftarrow \theta$ and $\psi_{before} \leftarrow \psi$;
  Clear the experience replay buffer D.
end while

4.3.2. *Algorithm description.* The DRL-TOSC algorithm, built on the DRL framework, dynamically optimizes medical task offloading and AI service caching by training a policy and critic network. Using the policy gradient method, it makes intelligent decisions to maximize system efficiency in MCPS. Algorithm 1 details its implementation.

a) Initialization phase (Line 1): Initialize the weights and biases of the policy network $\pi_{\text{after}}(a_t|s_t)$ and the critic network $V_\psi(s_t)$. Establish an experience replay buffer D to store agent-environment interaction data.

b) Experience data sampling (Line 2): Training utilizes experience data from the previous policy π_{before} to generate historical interaction records under the current policy.

c) Outer iteration loop (Lines 3-21): In Line 3, the algorithm begins training within a loop of predefined maximum iterations. In Lines 4-11, for each time slot t, it records the current state, generates task offloading and caching decisions, executes actions, computes rewards, records the next state, and stores experience data in replay buffer D. In Lines 13-20, it optimizes by sampling mini-batches from D, computing the advantage function Δ_t, and updating policy and critic network parameters using the Adam optimizer. Buffer D is updated and cleared. The loop completes a single training process in MCPS.

Complexity analysis: The complexity of the DRL-TOSC algorithm consists of two primary components: Neural network computations and mask generation/application in MCPS. The trained Actor network is deployed on the ES to facilitate real-time decision-making. The neural network consists of two hidden layers containing L_1 and L_2 neurons, respectively, with an input dimension of $6M$ and an output dimension of $3M + 2H$. The complexities of dynamic mask

generation and application are MH and M, respectively. Consequently, the overall algorithm complexity is expressed as $6ML_1 + L_1L_2 + (3M + 2H)L_2 + MH + M$.

TABLE 2. Experimental settings.

Parameters	Value
Learning rate	0.001
Discount factor γ	0.9
Energy consumption coefficient η	10^{-27}
Number of MDs	[50, 1000]
Number of time slots t	2^7
Noise power σ^2	2×10^{-13}
Communication bandwidth W	2 MHz
Storage capacity of ES	200 MB
Input/Output data size	[1, 5] MB
Number of AI services cached at the edge	[10, 100]
Local computing capability f_i^l	1 GHz
Computing capability of ES f_e	10 GHz
Transmission power of MD	1 W
Transmission power of ES	10 W
Data transmission rate to cloud	20 Mbps

5. Experimental results and discussion

This section presents the simulation setup, evaluates the convergence behavior of the proposed algorithm, and demonstrates its superiority over baseline methods through comparisons across multiple MCPS scenarios. The main experimental parameters are summarized in Table 2.

5.1. **Experimental Setup.** Our experiments are implemented using the PyTorch framework. The hardware platform includes an Intel(R) Core(TM) i9-13900F CPU, 32GB RAM, and an NVIDIA RTX 4060Ti GPU to accelerate model training and inference. The signal coverage range of ESs is randomly set between 100 and 350 meters. The types of requested AI services follow a Zipf distribution [24], where the probability of service q being requested by a MD at time slot t is given by:

$$p_q = \frac{\frac{1}{q^n}}{\sum_{q=1}^{Q} \frac{1}{q^n}}, \forall q \in Q, \tag{28}$$

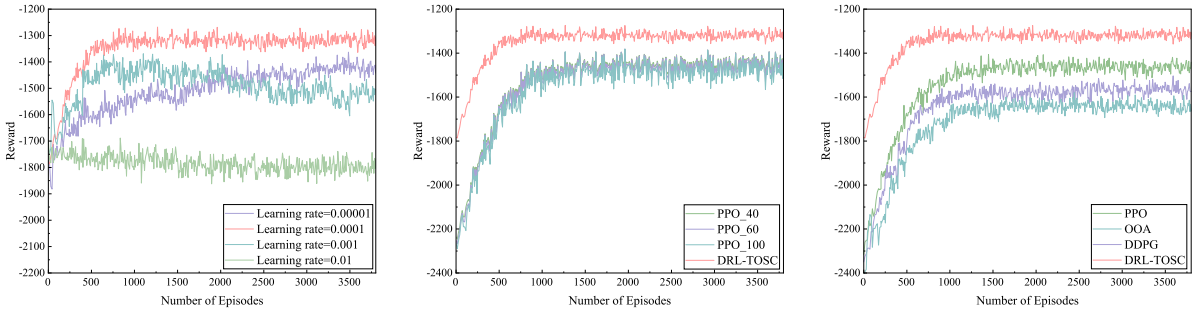
where the parameter $n = 0.8$ is used to reflect the imbalance in service demand [33].

The average channel gain is defined as:

$$\bar{h} = K \left(\frac{3 \times 10^8}{4\pi f_n d_i} \right)^e, \tag{29}$$

where K denotes the channel gain factor, e denotes the path loss exponent, f_n is the operating frequency, and d_i is the distance between device i and the ES. We follow the settings in [10,17,18] with $K = 4.11$, $f_n = 915$ MHz, and $e = 3$.

We set $\varepsilon_t = \varepsilon_e = 0.5$. For latency-sensitive tasks, the delay can be minimized by increasing the time weight ε_t and reducing the energy weight ε_e. Conversely, when the battery energy of the device is low, optimizing energy consumption can be achieved by decreasing ε_t and increasing ε_e in MCPS.



(A) (B) (C)

FIGURE 3. (a) The impact of different learning rates on the convergence of reinforcement learning training. (b) DRL-TOSC and PPO-based performance comparison with different penalty factors. (c) DRL-TOSC algorithm performance comparison with other benchmark algorithms.

The following benchmark algorithms are used to evaluate the effectiveness of DRL-TOSC.

(1) Local Execution Strategy (LES): Medical tasks are executed locally on MDs.

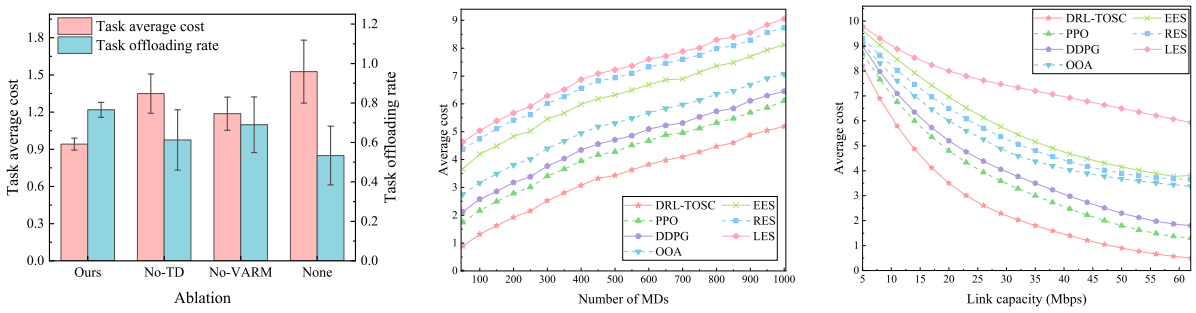
(2) Edge Execution Strategy (EES): Medical tasks are offloaded to ESs. If the ES does not cache the required AI service, it retrieves the service from the cloud.

(3) Random Execution Strategy (RES): The execution locationlocal, ES, or cloudis selected randomly according to the current cache state.

(4) Online Offloading Algorithm (OOA) [34]: A method originally designed for UAV-assisted MEC, which dynamically adjusts service caching based on user preferences while jointly optimizing trajectory planning and task offloading.

(5) PPO-based Execution Strategy [32]: A DRL-based strategy similar to the proposed scheme but without the VARM mechanism, where invalid offloading actions are penalized with high negative rewards.

(6) DDPG-based Caching and Offloading [31]: A DRL-based method that models service caching and task offloading as a long-term mixed-integer nonlinear program (MINLP) to optimize caching and computation jointly.



(A) (B) (C)

FIGURE 4. (a) Comparison of task average cost and offloading rate under different ablation settings. (b) Comparison of average costs across methods under different number of MDs. (c) Comparison of average costs under different link capacities.

To validate the superiority and adaptability of DRL-TOSC in MCPS, we conduct experiments on both convergence and performance. First, we examine its convergence speed and stability under different learning rates and compare it with PPO, DDPG, and OOA. Then, we evaluate its performance in terms of average cost, cache hit rate, offloading rate, and reward across diverse MCPS scenarios, including varying medical task loads, link capacities, server computational power, MEC storage sizes, and preference parameters.

5.2. **Convergence Analysis.** Fig.3(a) illustrates the effect of varying learning rates on the convergence of reinforcement learning in MCPS. A learning rate of 0.0001 achieves fast convergence and the highest stable reward, balancing exploration and convergence. A rate of 0.00001 converges slower and stabilizes at a lower reward, indicating prolonged training. A rate of 0.001 converges quickly but results in a lower final reward, while 0.01 causes high fluctuations and the lowest reward, suggesting instability. Thus, 0.0001 is selected for subsequent experiments in MCPS.

Fig.3(b) compares DRL-TOSC with a PPO-based medical task offloading and AI service caching scheme [32] under penalty factors of 40, 60, and 100. DRL-TOSC achieves faster convergence and consistently maintains the highest reward, demonstrating superior optimization for MCPS. In contrast, PPO shows similar convergence behavior across different penalty factor settings but exhibits slower convergence and lower final rewards than DRL-TOSC, indicating its limited capability in optimizing medical task offloading and service caching. As the penalty factor increases, PPO suffers from greater training fluctuations, while the marginal improvement in convergence suggests that its ability to constrain invalid offloading actions is weak. DRL-TOSC, on the other hand, effectively refines policies and significantly enhances system performance. Therefore, in subsequent experiments, the PPO penalty factor is fixed at 60 to balance convergence stability and training fluctuations in MCPS.

Fig.3(c) compares the convergence performance of DRL-TOSC, PPO, DDPG, and OOA during training in MCPS. DRL-TOSC demonstrates the fastest convergence in the early stages and maintains the highest reward throughout training, confirming its superiority in optimizing medical task offloading and AI service caching. In contrast, PPO, DDPG, and OOA converge more slowly and achieve lower final rewards. OOA applies a user preference-based dynamic update strategy to optimize UAV-assisted caching, trajectory planning, and task offloading. However, its lack of long-term accumulated reward optimization reduces adaptability to dynamic MCPS environments, limiting convergence performance. PPO extends OOA by introducing an invalid action penalty factor, which reduces policy randomness, accelerates convergence, and improves the final reward. Nevertheless, PPO still falls short of DRL-TOSC, as DRL-TOSC leverages the VARM mechanism to eliminate invalid actions and enhance both convergence quality and stability. DDPG suffers from noise sensitivity, leading to unstable gradients and inefficient exploration. Overall, DRL-TOSC consistently outperforms all baselines in convergence speed, final reward, and stability, validating its effectiveness in dynamic MCPS scenarios.

5.3. **Results Analysis and Discussion.** We further evaluate the proposed algorithm against baselines across diverse scenarios.

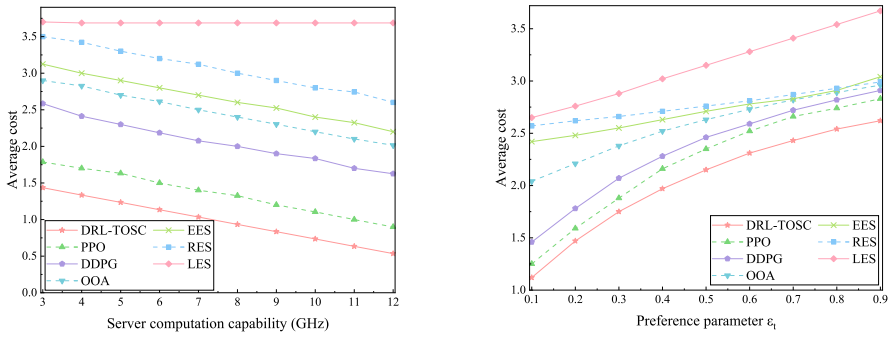
1) Average cost and offloading rate performance under different ablation settings: Fig.4(a) illustrates the impact of different ablation configurations on the average cost and offloading rate of medical tasks in MCPS. The full model ("Ours"), which integrates both TD learning and VARM, achieves the best performance with the lowest task cost and highest offloading rate. This confirms the critical contribution of both modules in improving decision quality and system efficiency for medical task offloading and AI service caching. When TD is removed ("No-TD"), the algorithm shows a clear increase in task cost and a decrease in offloading rate, highlighting TDs role in accelerating convergence and providing more accurate value estimates in dynamic environments. Eliminating VARM ("No-VARM") also leads to degraded performance, with higher task cost and a lower offloading rate. Without VARM, the policy space becomes cluttered with infeasible offloading actions, slowing down learning and increasing the likelihood of suboptimal decisions. The configuration without both TD and VARM ("None") performs the worst, with the highest average cost and lowest offloading rate. These results demonstrate that TD and VARM are complementary and essential, and that neither alone is sufficient to ensure robust learning and generalization in MCPS.

In conclusion, TD and VARM produce a synergistic effect within the DRL-TOSC framework: TD accelerates learning, while VARM improves policy validity. Together, they significantly reduce task execution cost and enhance offloading efficiency, underscoring their practical relevance in real-world MCPS scenarios.

2) Average cost performance under different numbers of MDs: Fig.4(b) compares the average cost performance of DRL-TOSC and various baseline algorithms under different medical task sizes in MCPS. As the number of MDs increases, the average cost for all algorithms rises, indicating that both computational and communication resource consumption grow with task complexity. DRL-TOSC consistently achieves the lowest average cost across all task sizes, benefiting from its dynamic offloading and caching adjustments, as well as the VARM mechanism that optimizes the action space, enhances learning efficiency, and reduces ineffective exploration. While PPO and DDPG perform better than baseline algorithms such as OOA, EES, and RES under smaller MD task loads, their costs increase significantly as the number of MDs grows, reflecting limited optimization ability under heavy workloads. OOA adjusts based on user preferences and makes decisions solely from the current environment, lacking long-term optimization capability. LES incurs the highest average cost due to limited local computing power; as the number of MDs increases, its latency and energy consumption rise sharply, demonstrating poor scalability. EES and RES, which lack adaptive offloading strategies, fail to adjust to varying medical task loads, leading to inferior performance compared to DRL-TOSC.

3) Average cost performance under different link capacities: Fig.4(c) compares the average cost performance of DRL-TOSC and benchmark algorithms under different link capacities in MCPS. DRL-TOSC achieves the fastest cost reduction and maintains the lowest cost at high link capacities (>40 Mbps), eventually stabilizing at 0.9, demonstrating its ability to efficiently utilize bandwidth resources and sustain long-term optimization advantages for medical task offloading and AI service caching. PPO and DDPG show rapid cost reduction in the 20-40 Mbps range, indicating a certain level of optimization under moderate link capacities. However, their curves plateau at higher capacities, reflecting limited exploration efficiency and an inability to further reduce costs. In contrast, OOA, EES, and RES achieve smaller cost reductions, suggesting that their fixed offloading strategies fail to fully exploit the benefits of increased link capacity. At low link capacities (<20 Mbps), their high costs highlight weaker adaptability in resource-constrained environments. The cost of LES remains largely unaffected by link capacity variations, as local execution on MDs is constrained by device capability, offering limited optimization potential. At a link capacity of 20 Mbps, DRL-TOSC reduces the average cost by 27.1%, 32.7%, 41.6%, 49.7%, 46.2%, and 56.3% compared to PPO, DDPG, OOA, EES, RES, and LES, respectively.

4) Average cost performance under different server computing capabilities: Fig.5(a) compares the average cost performance of DRL-TOSC and other benchmark algorithms under different server computing capabilities. DRL-TOSC consistently maintains the lowest cost, with the most significant reduction as computing capacity increases. In contrast, EES and RES lack dynamic server scheduling, limiting their optimization, while LES relies entirely on local computation, leading to nearly unchanged costs regardless of server performance. These results confirm that DRL-TOSC outperforms benchmark methods under varying server capacities, effectively adapting to network changes and reducing system costs.

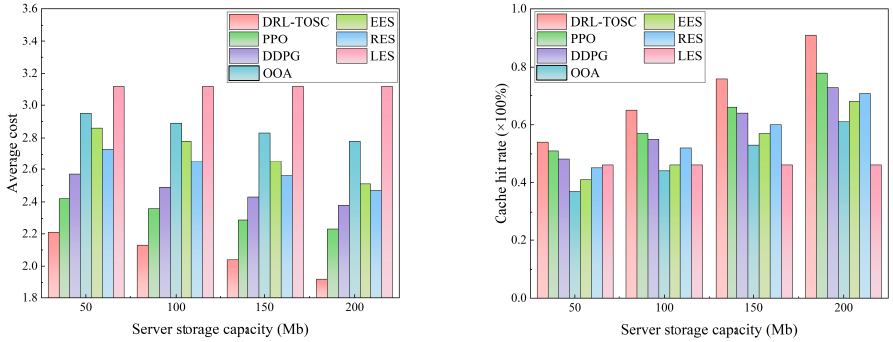


(A) (B)

FIGURE 5. (a) Comparison of average costs under different ES computing capabilities. (b) Comparison of average costs under different preference parameters.

5) Average cost performance under different preference parameters ε_t: Fig.5(b) compares the average cost performance of DRL-TOSC and other benchmarks under different preference parameters, where $\varepsilon_t + \varepsilon_e = 1$. As ε_t increases from 0.1 to 0.9 (with ε_e decreasing accordingly), the system becomes more latency-sensitive, prioritizing delay reduction at the cost of higher energy consumption. DRL-TOSC shows moderate cost growth, indicating strong adaptability to varying preferences. PPO and DDPG are more affected by changes in ε_t, while OOA, EES, and RES maintain consistently high costs with little impact from ε_t, reflecting their fixed strategies. LES incurs the highest cost, which increases sharply as ε_t rises, indicating that local computation becomes more burdensome under higher latency sensitivity.

6) Average cost performance under different MEC storage capacities: Fig.6(a) compares the average cost performance of DRL-TOSC and benchmark algorithms under different MEC storage capacities in MCPS. As storage capacity increases, task cache hit rates improve and execution costs decline, though the effects vary across algorithms. PPO and DDPG achieve notable cost reductions, indicating stronger adaptability to storage expansion. OOA, EES, and RES show limited gains, reflecting weaker utilization of additional storage resources. LES maintains the highest cost, demonstrating minimal sensitivity to storage variations due to the restricted computing capability of MDs. When storage increases from 50 MB to 200 MB, DRL-TOSC reduces its average cost from 2.21 to 1.92, achieving a 13.1% decrease. This confirms the advantage of DRL-TOSC in MCPS, where it coordinates medical task offloading and AI service caching by optimizing link loads, eliminating invalid actions, and refining offloading strategies.



(A) (B)

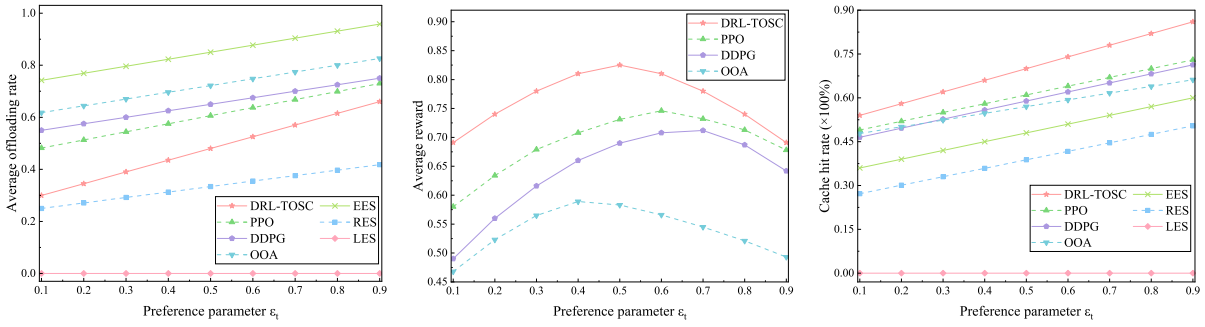
FIGURE 6. (a) Comparison of average costs under different MEC storage capacities. (b) Comparison of cache hit rates under different MEC storage capacities.

7) Cache hit rate comparison under different MEC storage capacities: Fig.6(b) compares the cache hit rates of DRL-TOSC and benchmark algorithms under different MEC storage capacities in MCPS. Cache hit rate is a key metric for evaluating AI service caching performance, as higher hit rates reduce medical data transmission overhead and improve system response times. As MEC server storage capacity increases, all algorithms achieve higher hit rates, though with significant differences. DRL-TOSC consistently maintains the highest hit rate across all storage conditions. For example, when storage capacity increases from 50 MB to 200 MB, DRL-TOSCs cache hit rate rises from 54% to 91%, an improvement of 40.7%. At 200 MB, DRL-TOSC outperforms PPO and DDPG by 16.6% and 24.7%, respectively. OOA, EES, and RES exhibit lower cache hit rates, reflecting inefficient utilization of storage resources and suboptimal scheduling of medical tasks. LES shows the lowest cache hit rate, remaining nearly constant regardless of storage capacity, since all tasks are executed locally on MDs without leveraging caching. In summary, DRL-TOSC effectively integrates medical task offloading and AI service caching, maximizing storage utilization while consistently maintaining low average costs and high cache hit rates across diverse MCPS scenarios.

8) Performance of the average offloading rate, average reward, and cache hit rate under different preference parameters ε_t: Fig.7(a) compares the average offloading rate of DRL-TOSC and other benchmark algorithms under different preference parameters ε_t in MCPS. Before $\varepsilon_t = 0.5$, the system prioritizes energy consumption by minimizing long-distance data transmission to reduce communication energy consumption. After $\varepsilon_t = 0.5$, the system shifts its focus towards reducing latency, leveraging the computational power of the MEC servers and the cloud to minimize local execution delays, thus increasing the offloading rate. As ε_t increases (i.e., with a greater emphasis on latency sensitivity), the slope of the offloading rate reflects the system's sensitivity to changes in "latency preferences". LES does not perform offloading, maintaining an offloading rate of zero. EES achieves the highest offloading rate but lacks flexibility in adjustment. PPO, OOA, and DDPG show gradual increases in the offloading rate, but their efficiency is slightly lower due to the absence of the VARM mechanism for eliminating invalid actions. DRL-TOSC, by incorporating the VARM strategy, quickly adapts the offloading policy to match medical task preferences, providing higher flexibility and better balancing the system's offloading performance in MCPS.

Fig.7(b) compares the average reward of DRL-TOSC with other benchmark algorithms, including PPO, DDPG, and OOA, under different preference parameters ε_t in MCPS. Higher values of ε_t prioritize latency optimization, while lower values emphasize energy consumption optimization. DRL-TOSC achieves the highest reward at $\varepsilon_t = 0.5$, indicating its ability to balance energy consumption and latency effectively. PPO reaches the highest reward at $\varepsilon_t = 0.6$, DDPG at $\varepsilon_t = 0.7$, and OOA at $\varepsilon_t = 0.4$. These results suggest that OOA is more suitable for energy-sensitive scenarios, while PPO and DDPG are more latency-sensitive, with a stronger focus on latency optimization over energy consumption in MCPS.

Fig.7(c) compares the cache hit rate of DRL-TOSC with other benchmark algorithms under different preference parameters ε_t in MCPS. As ε_t increases, the cache hit rate generally improves, and the system increasingly relies on edge or cloud computing resources to reduce local execution latency and enhance cache utilization. DRL-TOSC achieves the highest cache hit rate across all preference parameters, rising from 54% to 86%, demonstrating its superior performance in cache updating and service popularity tracking. PPO's cache hit rate increases from 49% to 73%, but its performance is limited due to the removal of the VARM mechanism and the high penalty applied to the reward function when invalid actions are selected. Although the cache hit rates of DDPG, OOA, and EES also improve, the increase is relatively modest. In particular, LES fails to effectively utilize cache resources in MCPS.



(A) (B) (C)

FIGURE 7. (a) Average offloading rate under different preference parameters ε_t. (b) Average reward under different preference parameters ε_t. (c) Cache hit rate under different preference parameters ε_t.

6. Conclusion and future work

To address the strong coupling between medical task offloading and AI service caching in MCPS, we propose DRL-TOSC, which formulates the problem as a Markov Decision Process (MDP) and introduces VARM to eliminate invalid actions and renormalize the probability distribution over valid offloading actions, thereby improving decision efficiency. By incorporating the TD method, the medical agent learns the optimal policy online in dynamic environments. The proposed framework also accounts for transmission overhead for data-intensive medical tasks and link load constraints, effectively reducing latency and energy consumption. Experimental results demonstrate that DRL-TOSC reduces the average system cost by 42.3% and increases the cache hit rate by 20.7% compared to benchmark algorithms.

Future work will investigate the scalability of DRL-TOSC in large-scale MCPS deployments and its generalization to diverse healthcare scenarios. Furthermore, integrating emerging technologies such as federated learning will enhance data privacy and collaborative performance.

References

[1] Ala'anzy, M. A., Zhanuzak, R., Akhmedov, R., Mohamed, N., Al-Jaroodi, J. Dynamic load balancing for enhanced network performance in ioT-enabled smart healthcare with fog computing, *IEEE Access*, Vol.12, No.1, 2024, pp.188957-188975.

[2] AlZubi, A.A., Al-Maitah, M., Alarifi, A. Cyber-attack detection in healthcare using cyber-physical system and machine learning techniques, *Soft Comput.*, Vol.25, No.18, 2021, pp.12319-12332.

[3] Ben-Ameur, A., Araldo, A., Chahed, T., Dán, G. Cache allocation in multi-tenant edge computing: An online model-based reinforcement learning approach, *IEEE Trans. Cloud Comput.*, Vol.13, 2025, pp.459-472.

[4] Bi, S., Huang, L., Zhang, Y.-J. Joint optimization of service caching placement and computation offloading in mobile edge computing systems, *IEEE Trans. Wirel. Commun.*, Vol.19, No.7, 2020, pp.4947-4963.

[5] Chen, F., Tang, Y., Wang, C., Huang, J., et al. Medical cyber–physical systems: A solution to smart health and the state of the art, *IEEE Trans. Comput. Soc. Syst.*, Vol.9, No.5, 2021, pp.1359-1386.

[6] Chen, X., Wang, P., Yang, Y., Liu, M. Resource-constraint deep forest based intrusion detection method in internet of things for consumer electronic, *IEEE Trans. Consum. Electron.*, Vol.14, No.8, 2025. pp.3373126.

[7] Chu, W.B., Jia, X.M., Yu, Z.W., Liu, J.C.S., Lin, Y. Joint service caching, resource allocation and task offloading for MEC-based networks: A multi-layer optimization approach, *IEEE Trans. Mobile Comput.*, Vol.23, No.4, 2023, pp.2958-2975.

[8] Dong, C., Sun, Y., Shafiq, M., Hu, N., Liu, Y., Tian, Z. Optimizing mobility-aware task offloading in smart healthcare for Internet of Medical Things through multiagent reinforcement learning, *IEEE Internet Things J.*, Vol.11, No.8, 2023, pp.13677-13691.

[9] Feng, C., Han, P., Zhang, X., Zhang, Q., Liu, Y., Guo, L. Dependency-aware task reconfiguration and offloading in multi-access edge cloud networks, *IEEE Trans. Mobile Comput.*, Vol.23, No.10, 2024, pp.9271-9288.

[10] He, X., Pang, S.C., Qiao, S.B., Gui, H.Y., et al. Real-time scheduling of CPU/GPU heterogeneous tasks in dynamic IoT systems: Enhancing GPU and memory efficiency, *IEEE Trans. Mobile Comput.*, Vol.25, No.1, 2026, pp.583-598.

[11] Hossain, M.S., Shorfuzzaman, M. MetaParkinson: A cyber-physical deep meta-learning framework for n-shot diagnosis and monitoring of parkinson's patients, *IEEE Syst. J.*, Vol.17, No.4, 2023, pp.5251-5260.

[12] Kazmi, S.M. Ahsan, Khan, Z., Khan, A., Mazzara, M., Khattak, A.M. Leveraging deep reinforcement learning and healthcare devices for active travelling in smart cities, *IEEE Trans. Consum. Electron.*, Vol.71, No.2, 2024, pp.4475-4486.

[13] Kuwar, V., Sonwaney, V., Upreti, S., Ekatpure, S.R., et al. Real-Time data analytics and decision making in cyber–physical systems, in *Navigating Cyber-Physical Systems with Cutting-Edge Technologies*, IGI Global Scientific Publishing, 2025, pp.373-390.

[14] Liu, H., Li, Y., Wang, S. Request scheduling combined with load balancing in mobile-edge computing, *IEEE Internet Things J.*, Vol.9, No.21, 2022, pp.20841-20852.

[15] Lv, Z., Chen, D., Lou, R., Alazab, A. Artificial intelligence for securing industrial-based cyber–physical systems, *Future Gener. Comput. Syst.*, Vol.117, 2021, pp.291-298.

[16] Mahmoodi, S.E., Uma, R.N., Subbalakshmi, K.P. Optimal joint scheduling and cloud offloading for mobile applications, *IEEE Trans. Cloud Comput.*, Vol.7, No.2, 2016, pp.301-313.

[17] Qiao, S.B., Guo, Q., Shi, F.D., Wang, M., et al. SIBW: A swarm intelligence-based network flow watermarking approach for privacy leakage detection in digital healthcare systems, *IEEE J. Biomed. Health Inform.*, Vol.30, No.3, 2026, pp.1912-1924.

[18] Qiao, S.B., Guo, Q., Wang, M., Zhu, H.H., et al. FRW-TRACE: Forensic-ready watermarking framework for tamper-resistant biometric data and attack traceability in consumer electronics, *IEEE Trans. Consum. Electron.*, Vol.71, No.3, 2025, pp.8234-8245.

[19] Salau, B.A., Rawal, A., Rawat, D.B. Recent advances in artificial intelligence for wireless internet of things and cyber–physical systems: A comprehensive survey, *IEEE Internet Things J.*, Vol.9, No.15, 2022, pp.12916-12930.

[20] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O. Proximal policy optimization algorithms, *arXiv preprint arXiv:1707.06347*, 2017.

[21] She, H., Yan, L.X., Guo, Y.A. Efficient end-edge-cloud task offloading in 6G networks based on multi-agent deep reinforcement learning, *IEEE Internet Things J.*, Vol.11, No.11, 2024, pp.20260-20270.

[22] Shishvan, O.R., Zois, D.-S., Soyata, T. Machine intelligence in healthcare and medical cyber–physical systems: A survey, *IEEE Access*, Vol.6, 2018, pp.46419-46494.

[23] Suura, S.R. Agentic artificial intelligence systems for dynamic health management and real-time genomic data analysis, *Eur. J. Anal. Artif. Intell.*, Vol.2, No.1, 2024.

[24] Tan, Z.H, Chen, B., Li, R.D, Jang, L., Xu, X.H. Efficient cache retrieving scheme in VANET via online reinforcement learning, *IEEE Trans. Veh. Technol.*, Vol.74, No.6, 2025, pp.9506-9519.

[25] Vijayalakshmi, P., Selvalakshmi, B., Subashini, K., Sudhakar, G., et al. Enhanced security framework with blockchain for Industry 4.0 cyber–physical systems, exploring IoT integration challenges and applications, *Artificial Intelligence and Machine Learning for Industry 4.0*, Wiley, 2025, pp.247-265.

[26] Wang, J.Y, Pan, J.L., Esposito, F., Calyam, P., et al. Edge cloud offloading algorithms: Issues, methods, and perspectives, *ACM Comput. Surv. (CSUR)*, Vol.52, No.1, 2019, pp.1-23.

[27] Wang, M., Gao, Z.Y, Wang, H.Y, Qiao, S.B, Wang, J.H. End-to-end latency upper bounds and service chain deployment algorithm based on industrial internet network, *Simul. Model. Pract. Theory*, Vol.138, 2025, pp.103031.

[28] Wang, M., Wang, H.Y., Qiao, S.B., Chen, J.W., et al. Heterogeneous system list scheduling algorithm based on improved optimistic cost matrix, *Future Gener. Comput. Syst.*, Vol.164, 2025, pp.107576.

[29] Wu, Y., Wu, J., Chen, L., Liu, B., Yao, M., Lam, S. Share-aware joint model deployment and task offloading for multi-task inference, *IEEE Trans Intell Transp Syst.*, 2024.

[30] Xu, C.F., Guo, J.X., Li, Y.P., Zou, H.D., et al. Dynamic parallel multi-server selection and allocation in collaborative edge computing, *IEEE Trans. Mobile Comput.*, Vol.23, No.11, 2024, pp.10523-10537.

[31] Xue, Z., Liu, C., Liao, C., Han, G., Sheng, Z. Joint service caching and computation offloading scheme based on deep reinforcement learning in vehicular edge computing systems, *IEEE Trans. Veh. Tech.*, Vol.72, No.5, 2023, pp.6709-6722.

[32] Yan, J., Bi, S., Zhang, Y. Offloading and resource allocation with general task graph in mobile edge computing: A deep reinforcement learning approach, *IEEE Trans. Wirel. Commun.*, Vol.19, No.8, 2020, pp.5404-5419.

[33] Zeng, J., Duan, J., Cao, W., Wu, C. Topics modeling based on selective Zipf distribution, *Expert Syst. Appl.*, Vol.39, No.7, 2012, pp.6541-6546.

[34] Zhang, J., Shen, Y., Wang, Y., Zhang, X., Wang, J. Dual-timescale resource allocation for collaborative service caching and computation offloading in IoT systems, *IEEE Trans. Ind. Inform.*, Vol.19, No.2, 2022, pp.1735-1746.
[35] Zhang, J.N., Chen, J.W., Bao, X., Liu, C.H., et al. Dependent task offloading mechanism for cloud-edge-device collaboration, *J. Netw. Comput. Appl.*, Vol.216, 2023, pp.103656.
[36] Zhang, W.Q., Zhang, G.L., Mao, S.W. Deep reinforcement learning based joint caching and resources allocation for cooperative MEC, *IEEE Internet Things J.*, Vol.11, No.7, 2023, pp.12203-12215.
[37] Zhao, L., Zhao, Z.J., Hawbani, A., Liu, Z., et al. Dynamic caching dependency-aware task offloading in mobile edge computing, *IEEE Trans. Comput.*, Vol.74, No.5, 2025, pp.1510-1523.
[38] Zhou, R.T., Huang, Y.F., Wang, Y.F., Jiao, L., et al. User preference oriented service caching and task offloading for uav-assisted MEC networks, *IEEE Trans. Serv. Comput.*, Vol.18, No.2, 2025, pp.1097-1109.



Sibo Qiao - received his Ph.D. degree from the China University of Petroleum, Qingdao, in 2023. He received his M.Sc. degree from the same University in 2020. Currently, he is working in the College of Software at Tiangong University. His research interests include task offloading and image processing.



Qin Xie - is currently pursuing her M.Sc. degree in Electronic Information at the School of Control Science and Engineering, Tiangong University, China. Her research focuses primarily on edge computing and task offloading multi-tier medical cyber-physical systems.



Min Wang - received her Ph.D. degree in Optical Engineering from Tianjin University in 2011 and is a senior visiting scholar at Virginia Tech. She is now a Professor in the College of Life Sciences at Tianjin Tiangong University. Her research interests include task offloading and resource allocation.



Xiao He - is currently pursuing his Ph.D. degree at China University of Petroleum (East China), and he received his M.Sc. degree from China University of Petroleum in 2023. His research interests include industry internet, edge computing, end-edge-cloud computing, and deep reinforcement learning in multi-tier medical cyber-physical systems.



Qifei Du - worked for Tianjin Inventec Electronic Technology Co., Ltd., where he was engaged in cloud storage embedded development. In 2010, he joined Tiangong University, where he has been working ever since. His main research interests include end-edge-cloud computing, task offloading and embedded development.



Mohammad Shabaz - is a Senior Assistant Professor at Marwadi University, India. His research includes multimedia-based emerging technologies, data analytics for neuroscience as a service (NaaS), and computational analysis.