Information Sciences 715 (2025) 122238

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Reward shaping in DRL: A novel framework for adaptive resource management in dynamic environments

Mario Chahoud [a,b], Hani Sami [c,b], Rabeb Mizouni [d], Jamal Bentahar [e,a,*], Azzam Mourad [e,b], Hadi Otrok [d], Chamseddine Talhi [b]

[a] Concordia Institute for Information Systems Engineering, Concordia University, Montreal, QC, Canada
[b] Artificial Intelligence & Cyber Systems Research Center, Department of CSM, Lebanese American University, Beirut, Lebanon
[c] Department of Software and IT Engineering, Ecole de Technologie Superieure (ETS), Montreal, Quebec, Canada
[d] Center of Cyber-Physical Systems (C2PS), Department of Computer Science, Khalifa University, Abu Dhabi, United Arab Emirates
[e] KU 6G Research Center, Department of Computer Science, Khalifa University, Abu Dhabi, United Arab Emirates

ARTICLE INFO

Keywords:
Resource management
Reinforcement learning
Reward shaping
On-demand
Dynamic environment
Fast convergence

ABSTRACT

In edge computing environments, efficient computation resource management is crucial for optimizing service allocation to hosts in the form of containers. These environments experience dynamic user demands and high mobility, making traditional static and heuristic-based methods inadequate for handling such complexity and variability. Deep Reinforcement Learning (DRL) offers a more adaptable solution, capable of responding to these dynamic conditions. However, existing DRL methods face challenges such as high reward variability, slow convergence, and difficulties in incorporating user mobility and rapidly changing environmental configurations. To overcome these challenges, we propose a novel DRL framework for computation resource optimization at the edge layer. This framework leverages a customized Markov Decision Process (MDP) and Proximal Policy Optimization (PPO), integrating a Graph Convolutional Transformer (GCT). By combining Graph Convolutional Networks (GCN) with Transformer encoders, the GCT introduces a spatio-temporal reward-shaping mechanism that enhances the agent's ability to select hosts and assign services efficiently in real time while minimizing the overload. Our approach significantly enhances the speed and accuracy of resource allocation, achieving, on average across two datasets, a 30% reduction in convergence time, a 25% increase in total accumulated rewards, and a 35% improvement in service allocation efficiency compared to standard DRL methods and existing reward-shaping techniques. Our method was validated using two real-world datasets, MOBILE DATA CHALLENGE (MDC) and Shanghai Telecom, and was compared against standard DRL models, reward-shaping baselines, and heuristic methods.

1. Introduction

In dynamic edge computing environments, efficiently managing computational resources is essential for selecting appropriate hosts and assigning services to meet user application demands. These environments are characterized by rapidly changing user locations and fluctuating service loads, which complicates resource allocation [1]. Mobility introduces additional challenges, as the

* Corresponding author.
E-mail address: jamal.bentahar@ku.ac.ae (J. Bentahar).

https://doi.org/10.1016/j.ins.2025.122238
Received 27 December 2024; Received in revised form 19 April 2025; Accepted 21 April 2025

Available online 24 April 2025

available resources and optimal host locations shift in real-time. A prime example can be seen in applications such as video streaming optimization and smart city infrastructure [2]. In video streaming, edge servers must dynamically assign computing resources to handle real-time video processing and delivery as users move across different regions, ensuring low latency and high-quality service. Similarly, in smart cities, services like traffic monitoring or public safety systems must deploy computational tasks to edge hosts, efficiently distributing workloads based on the mobility and concentration of users [3]. Unlike static environments where resource allocation strategies can be predefined, real-world edge computing environments are highly dynamic, requiring continuous adaptation. As user demand fluctuates and mobility affects resource availability, optimizing computational resource allocation becomes crucial for sustaining service quality and avoiding overloading any single node. Therefore, an adaptive, learning-based framework is required to dynamically optimize resource allocation while maintaining real-time responsiveness and system efficiency.

Deploying and managing computational resources in real-time are particularly challenging in dynamic environments where user demands and mobility patterns change frequently. The on-demand architecture for resource management, which utilizes Docker containers managed by Kubernetes masters, provides a flexible solution for deploying edge services on volunteer host devices [4]. This architecture transforms these devices into small, localized servers, allowing computational resources to be deployed closer to users, improving latency and service responsiveness. This setup allows rapid adjustments in response to changing demands. However, it still lacks an essential optimization component—deciding where and how services should be deployed in real time [5]. For applications like autonomous vehicles or Augmented Reality systems (AR), where real-time data processing is critical, effective resource management requires not just on-demand deployment but also an intelligent mechanism to optimize host selection and service allocation. Without this optimization layer, on-demand architectures alone cannot fully address the complexities introduced by constant changes in user mobility and computational needs, leading to latency, suboptimal resource utilization, and potential service failure during high-demand periods.

Numerous methods have been introduced for resource management, each offering distinct strengths and facing specific limitations. Traditional static resource allocation methods consistently struggle to adapt quickly to changes in demand, resulting in inefficiencies and potential service interruptions [6]. Heuristic approaches, while faster, often lack the precision necessary for complex, dynamic environments [7]. On the other hand, Deep Reinforcement Learning (DRL) has emerged as a promising solution for dynamic resource management due to its ability to learn from interactions with the environment and adapt over time using a reward function [8]. The use of DRL is vital in this scenario as it can optimize decision-making in real-time, allowing systems to respond to varying demands and changing conditions effectively [9]. However, many existing DRL-based methods face significant challenges when operating in dynamic environments, specifically when the group of devices requesting services exhibits mobility factors, which complicate achieving critical goals such as fast convergence, higher rewards, and improved accuracy in decision-making. These DRL solutions frequently neglect to incorporate the inherent complexities of dynamic environments into their frameworks [10]. While the existing solutions demonstrate accuracy for a small-time step, their decision-making will not be effective as the environment evolves [11]. This limitation reduces their ability to provide optimal resource management in scenarios where timely adjustments are crucial. Consequently, overcoming these DRL limitations is essential to boost responsiveness and efficiency in resource allocation, thereby improving service delivery across diverse applications.

Once an effective DRL solution is established, a second challenge arises which is the speed of convergence when the environment undergoes changes [12]. The speed of convergence in this context refers to the speed of the learning algorithm when reaching an optimal solution. Existing DRL-based methods often suffer from prolonged learning times, which can lead to suboptimal policy development, especially in rapidly changing environments [13]. While DRL can optimize decision-making under certain conditions, its effectiveness reduces when there is a need to adapt quickly to new dynamics. These methods frequently struggle to manage the rapid changes in user location and demand, resulting in delayed or suboptimal decisions [14]. Therefore, it is essential to improve DRL solutions not only to enable effective decision-making but also to accelerate convergence, ensuring optimal resource management in dynamic environments.

To tackle the decision-making challenges that arise when users change locations, we utilize Proximal Policy Optimization (PPO) within a novel Markov Decision Process (MDP) framework. This approach enables effective real-time optimization of resource allocation decisions. By integrating PPO, the DRL agent can adapt to the dynamic nature of user mobility [15]. The MDP framework accounts for critical factors such as the device computational limits, the distance between users and resources, and the variations in user mobility. This comprehensive approach ensures that resources are allocated efficiently and responsively, enhancing overall service delivery in a dynamic environment [16].

In parallel, there has been a rise in the application of reward shaping in reinforcement learning [17]. Reward shaping accelerates convergence and enables the system to adapt quickly to evolving conditions, ensuring continuous and reliable service in dynamic environments. By leveraging potential-based reward shaping, which enhances the learning process without altering the optimal reward function policy, more informative feedback is provided to the DRL agent. The use of potential-based reward shaping ensures faster learning and more accurate decision-making, despite significant environmental changes. To address the challenge of adapting quickly to changes in user groups, we incorporate reward shaping into our DRL framework, which delivers enhanced feedback signals to the DRL agent, expediting the learning process. For the first time, we are integrating Graph Convolutional Networks (GCN) and transformer-based encoders within a DRL framework named the Graph Convolutional Transformer (GCT). GCN is essential for analyzing spatial dependencies, enabling the model to understand and utilize information from neighboring states. This capability is essential in dynamic environments, as it helps the agent recognize how changes in one area can impact others, mimicking the way it transitions from one state to another in response to spatial dynamics. Conversely, transformers are used to analyze temporal dependencies, allowing the model to track and adapt to changes over time [18]. By processing sequences of events, transformers ensure that the DRL agent considers the evolution of user behavior and environmental conditions, enhancing its responsiveness to temporal

shifts. Unlike conventional DRL methods that struggle with slow adaptation, our framework utilizes a structured spatio-temporal reward shaping mechanism that guides learning more efficiently. By leveraging Krylov subspace approximations, our approach reduces computational complexity while maintaining robust decision-making capabilities. We efficiently approximate transition probabilities within the Krylov subspace, significantly reducing computational complexity. To address the issues of slow learning and poor adaptability in dynamic edge computing scenarios, our framework incorporates Graph Convolutional Networks, Transformer encoders, and the Augmented Krylov Subspace Approximation. Together, these elements improve the agent's capacity to understand spatial patterns, track temporal dynamics, and estimate transition probabilities with greater computational efficiency. This is the first work to combine reward shaping with GCN and Transformer encoders in the context of dynamic resource management, bridging the gap between reinforcement learning convergence and real-world deployment in edge environments.

Our approach outperforms several existing methods, including greedy and heuristic strategies, DRL techniques that lack reward shaping, and those that combine GCN with RNNs for reward shaping in gaming environments, which are generally less complex than dynamic environments. These were selected based on their widespread use in dynamic environments and their relevance to edge computing applications. Our extensive experiments validate the effectiveness of our framework, demonstrating on average across two datasets a 30% reduction in convergence time, a 25% increase in total accumulated rewards, and a 35% improvement in service allocation compared to standard DRL and heuristic methods. This performance has been validated using real-life datasets, demonstrating that our method has the ability to quickly adapt to changes in user requests within dynamic environments, even as users move, while achieving higher reward values. To our knowledge, this work is the first to address reward shaping within the context of on-demand deployment architecture through DRL in resource management environments, which not only guarantees faster adaptation to dynamic changes in the environment, but also achieves more accurate decisions measured by higher reward.

The key contributions of this work are as follows:

1. Proposing a DRL framework that integrates for the first time reward shaping in the context of resource management that guarantees more accurate decisions and faster adaptation to environmental changes.
2. Introducing a novel MDP design for the DRL agent that considers the dynamic nature of users in terms of mobility as well as their demand.
3. Presenting a novel reward shaping mechanism with a unique shaping function, incorporating GCN and transformer encoders specifically for this purpose, a combination that has not been previously explored.

2. Literature review

This section provides an overview of the solutions proposed by researchers for dynamic resource management environments, along with the various reward-shaping techniques highlighted in the literature.

2.1. Resource management solutions in dynamic environments

Recent advancements in resource management across computing paradigms address critical challenges posed by the rapid growth of Internet of Things (IoT) applications and future network systems. In this context, [19] presents a monitoring and management system for microservices that enhances efficiency and Quality of Service (QoS) in Amazon Web Services (AWS) environments by predicting load variations and resource requirements. In the realm of 5G and beyond, [20] proposes a multi-objective approach for traffic forecasting using Deep Convolutional Neural Networks and Long Short-Term Memory networks, combined with the Tasmanian Devil Optimization-Elite method to improve precision and resource efficiency. Meanwhile, [21] suggests a Cluster-based Parallel Split Learning approach to reduce training latency by parallelizing within device clusters and optimizing resource management. Additionally, [22] reviews machine learning techniques in fog computing, emphasizing strategies for provisioning, scheduling, and load balancing, and highlighting future research directions. In dynamic environments, Researchers in [23] introduce the DetFed framework, which integrates federated learning with Time-sensitive Networks to enhance learning accuracy and network performance through dynamic resource scheduling. Additionally, [24] addresses cloud load balancing by integrating RL, large language models, and edge intelligence, improving throughput, security, and scalability while tackling challenges related to proprietary cloud APIs and multi-cloud interoperability. Moreover, in dynamic environments, the authors of [25] introduce a heuristic approach leveraging a Memetic Algorithm for service scheduling, with a focus on optimizing resource management objectives.

RL research has also evolved, with [26] exploring fog computing's role in managing time-sensitive requests through a framework using Software Defined Networking and advanced RL to optimize resource allocation for low-latency requirements in heterogeneous IoT environments. Moreover, [27] introduces the OSCAR model to enhance QoS in fog-assisted cloud computing by optimizing task scheduling and resource allocation using DRL, achieving better resource utilization and reduced response times. In addition to that, [28] presents solutions for intelligent resource management by leveraging DRL to optimize resource allocation while having dynamic service demands.

Both heuristic and DRL methods often fall short in dynamic environments due to their limited ability to quickly adapt to rapid changes. Heuristics lack real-time flexibility, while traditional DRL may suffer from slow and unstable learning, making them less effective in fast-changing conditions. As a result, these methods may fail to deliver the necessary responsiveness and efficiency required for robust performance in highly dynamic scenarios.

Table 1
Comparison of Existing Approaches.

Method	Adaptability	Complexity	Conv. Speed	Scalability
Heuristic [21,34,25]	Low	Low	Fast	Limited
DRL [26–28]	Mod	High	Slow	Mod
Reward Shaping [30–33]	Mod	Mod	Fast	Limited

2.2. Reward shaping methods

In the context of reward shaping within RL, [29] reviews the evolution of reward function design, focusing on reward shaping, intrinsic motivation, and related approaches. In [30], the authors propose an adaptive reward-shaping technique using a bi-level optimization approach, demonstrating improved performance in sparse-reward environments. Authors of [31] present Exploration-Guided Reward Shaping, a self-supervised framework that enhances learning in sparse-reward environments by combining intrinsic rewards with exploration bonuses. Moreover, [32] introduces a Bayesian reward-shaping framework incorporating prior beliefs to accelerate learning in the presence of delayed rewards, showing faster task completion with intuitive priors. Furthermore, [33] addresses low-speed convergence in RL by proposing a novel reward-shaping scheme integrating Graph Convolutional Recurrent Networks (GCRN), augmented Krylov basis, and look-ahead advice. This approach combines GCN and Bi-Directional Gated Recurrent Units (Bi-GRUs) to handle spatial and temporal dependencies more effectively. The Krylov basis provides accurate transition matrix estimation, while look-ahead advice refines reward shaping by considering both states and actions. Evaluations on Atari 2600 and MuJoCo demonstrate that this method accelerates learning and achieves higher rewards compared to existing GCN-based approaches.

A variety of approaches has been explored for dynamic resource management, spanning from traditional heuristics to modern AI-driven solutions. Heuristic-based methods [21] improve efficiency but often lack adaptability to real-time fluctuations. More recently, DRL-based approaches [28] have emerged as a promising solution, offering adaptive decision-making capabilities in highly dynamic environments. However, existing DRL techniques often suffer from slow convergence, high sample complexity, and sensitivity to environmental shifts, particularly when user mobility is involved. Moreover, it faces several challenges in scalability and computational feasibility. Standard RL models, such as DQN and PPO, require extensive training data and struggle with dynamic environmental shifts. Additionally, reward-shaping techniques [33] have been proposed to accelerate learning, yet their integration in dynamic edge computing scenarios remains limited. This paper builds upon these prior works by integrating GCN and Transformer encoders for a more structured reward-shaping mechanism, specifically tailored for dynamic environments with high user mobility. Unlike conventional DRL-based approaches, our proposed framework maintains the same fundamental resource requirements while significantly reducing convergence time, leading to lower overall data usage, computational overhead, and retraining costs. This efficiency makes our approach more suitable for real-world edge computing applications, where adaptive learning is necessary for dynamic service allocation in mobile environments.

To provide a structured comparative analysis of existing approaches, we summarize key aspects of heuristic-based, reinforcement learning, and graph-based optimization techniques in Table 1. This comparison highlights the limitations of traditional methods in terms of adaptability, computational feasibility, and convergence speed. While our approach builds on reward shaping methods, it introduces key enhancements to improve adaptability and scalability in dynamic, mobility-driven environments. By incorporating a Graph Convolutional Transformer (GCT), our framework captures spatial-temporal patterns more effectively, enabling faster learning and more efficient decision-making. In contrast to traditional DRL methods that struggle with slow convergence and high complexity, our structured reward shaping and message-passing mechanisms help the agent adapt smoothly to changing conditions with minimal computational overhead. Although the integration of GCNs, Transformers, and Krylov-based approximations may require more advanced deployment infrastructure compared to simpler DRL or heuristic approaches. However, our design was intentionally optimized to reduce overhead. These strategies help preserve computational efficiency while maintaining adaptability to dynamic environments, making the approach practical and scalable for real-time edge deployment.

3. Architecture

This section offers an in-depth explanation of the architecture shown in Fig. 1, detailing the key components and their interactions. It discusses the reasoning behind design decisions, the flow of data between modules, and how the architecture ensures flexibility and the efficiency of both individual components and the system as a whole.

3.1. Architecture overview

At its core, the on-demand architecture [35] is designed for efficiently deploying services close to the end-user. This proximity is achieved by deploying Docker containers on devices located near the requesting users. By utilizing these devices as “near edge” nodes, the architecture can process requests locally, thereby maximizing the QoS. The management of these containers is orchestrated by master nodes, which are responsible for maintaining the kubeadm cluster. This cluster serves as the backbone of the on-demand architecture, ensuring that the containers are connected and managed effectively. The architecture features a decision module within the master nodes that determines when to push services based on request volume. The cloud processes service requests and forwards them to the master nodes, which then allocate these services to the worker nodes. This would reduce the load on the cloud, and also

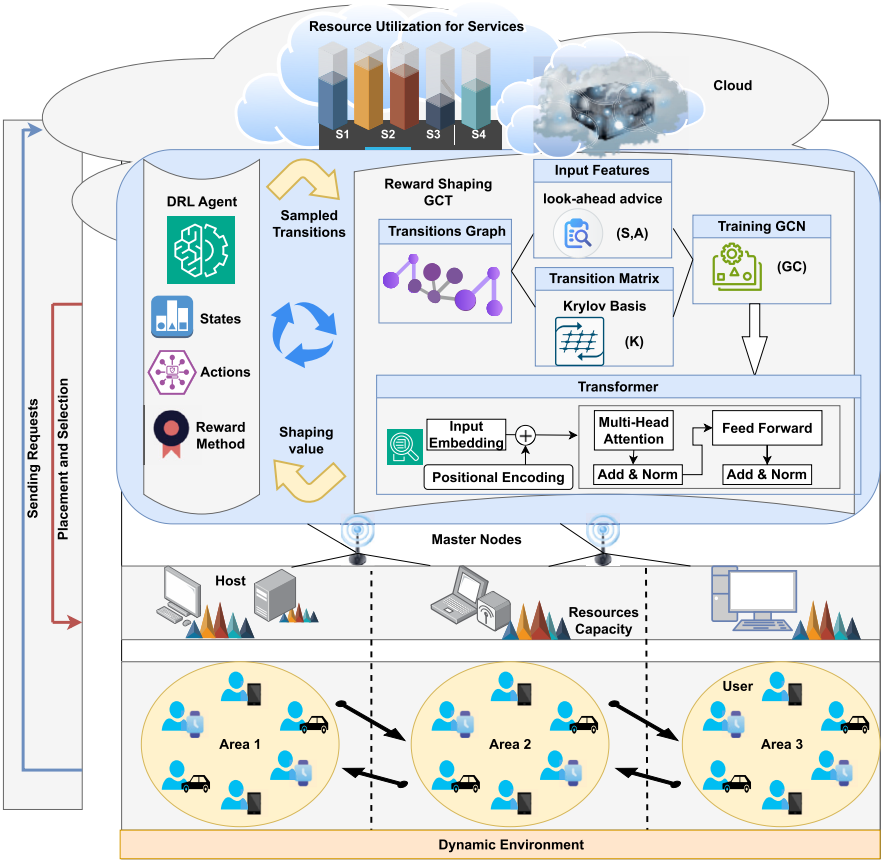

Fig. 1. The Overall Proposed Architecture.

decrease the number of users the cloud has to serve. The host devices are distributed across different geographical areas, and the on-demand architecture ensures that they are deployed and positioned close to the users who need services.

The deployment and placement of services and host devices are optimized through a DRL agent. This agent is tasked with maximizing specific objectives, as discussed in Section 4, while adhering to various constraints. The dynamic nature of the environment, characterized by constantly shifting user demands and varying locations of users in different areas, adds significant complexity to this process. Our framework optimizes resource placement to reduce contention and support effective decision-making. While the on-demand architecture handles host deployment, the DRL agent is tasked with placing services on available hosts based on user mobility and demand. By accounting for users' movement patterns in its optimization goals, the agent can adapt more accurately to changing conditions, ensuring responsive and efficient resource management. Future work may explore incorporating network latency models to further enhance deployment strategies in latency-sensitive environments.

Moreover, in highly dynamic environments, where the conditions change frequently, the DRL agent faces the challenge of learning and adapting to new data in real time. If the agent's learning process is prolonged, it may fail to converge before the environment changes again, rendering its learned policies ineffective. This introduces a continuous learning loop, where the agent must adapt to an ever-evolving environment. The risk here is that if the agent takes too long to converge, its policy may become obsolete, leading to inefficiencies. To tackle this challenge, we have integrated a reward-shaping method into the architecture. We are using a potential-based reward shaping to improve the learning speed of our reinforcement learning agents while preserving the optimal policy. This approach adjusts the reward signal by incorporating information from a potential function that reflects the state space. By enhancing the reward with this additional guidance, the agent learns more efficiently. Importantly, potential-based reward shaping ensures that, despite these adjustments, the agent still learns the same optimal strategy as it would without the shaping. This approach accelerates the learning process and helps the agent achieve higher reward functions in fewer episodes by providing more effective guidance in dynamic environments. This method enables the agent to quickly adapt to environmental changes, outperforming traditional learning methods. The DRL agent initiates the learning process. Subsequently, the second component, the reward-shaping module (GCT), processes these transitions to extract a shaping value that is incorporated into the overall reward. Once this module has matured, it can efficiently manage service placement and deployment, even in environments characterized by dynamic conditions and mobile users.

A key feature of our framework is its ability to preserve continual learning despite evolving environmental conditions. By utilizing a replay buffer, we ensure that past experiences are retained, preventing catastrophic forgetting while incorporating newly observed

changes. This balances short-term learning updates with long-term stability, ensuring that the agent remains responsive to mobility-driven fluctuations in service demand while maintaining an optimized policy. Unlike conventional approaches that may struggle with ongoing adaptation, our DRL-based framework naturally integrates continuous learning, making it well-suited for real-world, highly dynamic environments.

3.2. Architecture components

3.2.1. DRL environment

The DRL environment is structured as an MDP, which is represented by the following tuple $< S, A, P, r, \gamma >$. To build the DRL agent, these components need to be carefully designed according to our problem; mobile client selection and service assignment. The MDP tuple definition is presented in Section 4. The set S represents all possible states of the environment, each capturing a specific configuration or condition at any given time. The set A includes all possible actions the agent can undertake from any state, dictating how the agent interacts with the environment. The transition probability matrix P defines the likelihood of transitioning between states given an action, encapsulating the dynamics of the environment. The reward function r assigns a value to state-action pairs, providing feedback on the desirability of actions and guiding the agent's learning process. Finally, the discount factor γ determines the weight of future rewards compared to immediate rewards, influencing the agent's decision-making strategy. Together, these components form a comprehensive model for evaluating and optimizing strategies in complex and evolving scenarios. This formulation focuses on optimizing the number of successfully fulfilled requests, minimizing the distance between the users and the edge devices serving their requests, primarily in dynamic and mobile environments, and optimizing the number of deployed hosts while respecting service priorities. In the remainder of this section, the description of addition components from the proposed architecture is presented. Details about the formulated MDP are provided in Section 4.

3.2.2. Look-ahead advice

This mechanism uses both states and actions to improve the learning process by informing the agent of what happens next [36]. This helps the agent in performing improved decisions. By considering possible future changes, the agent can adapt more quickly and develop stronger strategies.

The primary objective of reward shaping is to refine and guide the reward function. This is achieved by incorporating scaling values from the shaping function F defined in Equation (1) into the reward function r (from the MDP tuple), to form the combined reward function R defined in Equation (2). By appending these values, the reward function is effectively directed and adjusted, resulting in an enhanced formulation.

$$F(S_t, S_{t+1}) = \gamma\phi(S_{t+1}) - \phi(S_t) \tag{1}$$

$$R(S_t, A_t, S_{t+1}) = r(S_t, A_t) + F(S_t, S_{t+1}) \tag{2}$$

where F is the shaping function, ϕ is the potential shaping functions that output a scalar value, S_t is the state representation at timestep t, and A_t is the action taken at timestep t.

In our proposed architecture, we present the state transitions through actions by a graph, where nodes are states and edges correspond to actions. Using the graph structure, we are able to capture spatial and temporal dependencies between the various states, thus extracting useful insights to control the reward signal. By applying the **message passing** technique commonly used in Hidden Markov Models (HMM) [37], we can calculate the probability that the agent is following an optimal trajectory based on the current state and action. This probability serves as a valuable signal for accelerating learning.

A more comprehensive approach involves constructing reward shaping based on both states and actions, which we adapt when building the proposed potential-based reward shaping solution. Building on the look-ahead advice outlined in [36], the shaping function is formulated to incorporate this dual consideration, resulting in the following structure:

$$F(S_t, A_t, S_{t+1}, A_{t+1}) = \gamma\phi(S_{t+1}, A_{t+1}) - \phi(S_t, A_t) \tag{3}$$

$$R(S_t, A_t, S_{t+1}, A_{t+1}) = r(S_t, A_t) + F(S_t, S_{t+1}, A_{t+1}) \tag{4}$$

As shown in both equations, the shaping function is a function of both the state and action with the intention of giving the DRL agent a more informed advice specific to a given action instead of the whole state. The message passing process involves computing forward and backward messages, which can be computationally intensive, especially when dealing with a large graph comprising numerous states and transitions [37]. The shaping function dynamically adjusts based on the agent's real-time transitions, ensuring that reward adjustments remain aligned with the evolving mobility patterns and service requests. Therefore, we present in the next section a graph neural network model that is well suited to perform message passing.

3.2.3. GCT model

This component operates through the following process: for each episode in the environment, we begin by collecting a sample of the agent's transitions. With these transitions, we construct a sub-graph, highlighted in Fig. 1. This sub-graph serves as the foundation for the next steps. The graph of states and action are used to form as an input for the GCN. Considering actions in addition to the state reinforces the look-ahead advice mechanism.

In its nature, a forward pass in GCN requires the adjacency matrix in the graph, which corresponds to the nature of connections for given graph as input. In our case, the provided graph represents the states transitions through actions. Therefore, the adjacency matrix should correspond to the probability transition matrix P from the MDP tuple. Since it is impossible to model the transition matrix due to the high dynamicity a non-stationarity in the environment, the proposed approach utilizes an approximation of P. Hence, we utilize the augmented Krylov algorithm to build the Krylov subspace K, which approximates the transition matrix P. The advantage of using the Krylov subspace lies in its ability to efficiently capture the dynamics of the environment with reduced computational complexity, making it well-suited for large-scale problems. The augmented Krylov algorithm requires only a subset of P to build the approximation. Following training on various versions of the graph depending on the DRL agent's experiences, the GCN produces an output denoted as (GC) for time step t. Before producing the shaping value out of GCN, a transformer encoder is appended to GCN to study the long term dependencies between the different states and actions of the given graphs. The overall network combining GCN and the transformer encoder is named GCT, which forms our shaping function.

GCN: In this model, the GCN handles the message passing by propagating information about rewarding states to neighboring nodes using a filter matrix (K representing the approximated P), while the transformer predicts the next reward shaping value. The GCT model effectively captures the intricate spatio-temporal dynamics within the environment by harnessing the complementary capabilities of GCN and transformer encoders. GCNs, due to their recursive design, are well-suited for message passing, allowing efficient propagation of information across the graph. This is supported by the use of the graph Laplacian, which acts as a filtering tool during the process. The Laplacian either represents the relationships and connectivity between nodes or serves as an approximation of the transition matrix, allowing the GCN to process complex graph-based data. Previous research, such as in [17], has employed the graph Laplacian in the context of reward shaping within GCN, under the assumption that the value function is smooth across the MDP graph. However, this smoothness assumption can introduce inaccuracies in the value function's approximation. To address these potential errors, the Krylov basis [38] is computed, offering a more precise approach to approximating the transition dynamics. Using the Krylov basis generated by the augmented Krylov algorithm generates a closer estimate to P, which is the actual transition matrix, compared to the graph Laplacian [39].

Initially, the GCN is employed to conduct semi-supervised learning, leveraging its recursive architecture and ability to propagate information from labeled data to neighboring nodes. By converting samples of the agent transitions into a graph structure, we can utilize the GCN to execute message passing. A forward pass in the GCN with two layers can be expressed as follows:

$$\phi_{GCN}(X) = \text{Softmax}(K\text{ReLU}(KXW_1)W_0) \tag{5}$$

Where the activation functions used are ReLu and Softmax, K is the GCN filter, X is the input matrix (states and actions forming a sub-graph), and W_0, W_1 are the layers' weights.

Transformer: In prior research, GCN was combined is RNNs to predict shaping values for reward shaping in Atari games [33], leveraging their sequential data processing capabilities. However, in the context of resource management within dynamic environments, where states and actions are represented as a graph and integrated with GCN, **transformer encoders** offer distinct advantages over RNNs. Unlike RNNs, which process data sequentially and may struggle with long-range dependencies due to their inherent recurrence, transformer encoders utilize a self-attention mechanism, allowing them to efficiently capture both local and global dependencies in the graph. This capability is especially useful in dynamic environments, where rapidly changing state-action relationships require full graph context for accurate shaping value prediction. By optimizing with attention mechanisms, this integration enhances the architecture's ability to handle complex scenarios effectively.

The transformer encoder inputs sequences using the self-attention mechanism and feed-forward neural network. For an input sequence X with embedding x_i, the encoder layer applies self-attention followed by a feed-forward network. The combined equation for one layer of the Transformer encoder is:

$$Attention(X) = Softmax(\frac{XW^Q(XW^k)^T}{\sqrt{d_k}})(XW^v) \tag{6}$$

$$Norm_{att} = LayerNorm(X + Attention(X)) \tag{7}$$

$$FFN(Norm_{att}) = max(0, Norm_{att}W_{1'} + b_1)W_{2'} + b_2 \tag{8}$$

$$Output = LayerNorm(Norm_{att} + FFN(Norm_{att})) \tag{9}$$

where W^Q, W^k, W^v are the weight matrices for the query, key, and value projections. Moreover, d_k is the dimension of the key vectors. Furthermore, $FFN()$ method is the Feed-Forward method, while $W_{1'}$ and $W_{2'}$ are the weight matrices for the feed-forward network. In conjunction with this, b_1 and b_2 are the biases for the feed-forward network. Lastly, $LayerNorm$ denotes layer normalization. In summary, each layer of the Transformer encoder applies self-attention followed by a feed-forward network, with residual connections and layer normalization applied at each step to ensure stable training and efficient representation learning.

Our reward shaping module, implemented using a GCN, is executed in parallel with the DRL agent PPO. The GCN is updated at predefined intervals, ensuring that it does not interfere with the real-time decision-making of the DRL agent. Additionally, our approach leverages an offline pre-training phase where the GCN learns spatial-temporal relationships from historical data. This structured learning phase significantly reduces the computational burden during deployment, as the DRL agent starts with an optimized policy, minimizing unnecessary updates. This dual-phase approach ensures that our solution maintains computational efficiency while effectively handling dynamic service allocation in mobile edge computing. To effectively handle the challenges of slow convergence and inefficient adaptation in dynamic edge computing environments, our framework integrates GCN, Transformers, and Augmented

Krylov Subspace Approximation. Together, these components help the agent capture spatial patterns, model long-term dependencies, and approximate transitions efficiently. By combining Krylov approximation with real-time GCN updates and Transformer-based temporal modeling, the system adapts well to non-stationary edge environments and maintains reliable decision-making under changing conditions.

To further enhance scalability, we optimize message passing and self-attention mechanisms by limiting computations to sampled transitions rather than maintaining a large, persistent graph structure. Our Transformer-based learning mechanism ensures that only the most relevant information is used for decision-making, minimizing unnecessary computations while preserving learning efficiency. This approach ensures that our framework remains scalable and suitable for dynamic edge computing environments without incurring significant performance strain.

3.2.4. Users

In this architecture, **users** request services, given that the number of users in different areas changes dynamically. Their interactions are deeply embedded within the environment, with their movements and activities meticulously tracked and recorded. These interactions are not just passive; they actively influence the environment's complexity and adaptability. By incorporating real-time user activities and varying user numbers, the environment becomes highly responsive to rapid changes in user behavior. This design ensures that the environment can simulate the dynamic shifts and challenges associated with fluctuating user demands, providing a realistic and robust setting for testing and optimizing the architecture's performance.

4. Model formulation and solution

In this section, we outline the design and setup of our components, including the MDP environment formulation, as well as the objective functions and constraints. Besides, we describe the PPO algorithm employed and the reward-shaping method to achieve rapid adaptation and higher reward in dynamic environments. Our framework operates in a non-tabular state-action space.

4.1. MDP formulation

Our architecture is designed to accommodate users moving across various locations, dynamically requesting services that need to be efficiently managed and deployed. The system is structured with a set of services $Sv = [Sv_1, Sv_2, \dots, Sv_n]$, where n denotes the total number of services. To support these services, we have m host devices, represented as $H = [H_1, H_2, \dots, H_m]$, each capable of hosting one or more services. Each host H_j is characterized by its resource capacity, detailed as $H_j = [H_{j,cpu}, H_{j,memory}, H_{j,diskspace}, H_{j,location}]$ representing the available CPU, memory, disk space, and geographic location of the host, respectively. Similarly, each service Sv_i has specific resource requirements to operate effectively, represented as $Sv_i = [Sv_{i,cpu}, Sv_{i,memory}, Sv_{i,diskspace}, Sv_{i,priority}]$. The $Sv_{i,priority}$ parameter indicates the priority level of a service, ranging from 0 (low priority) to 1 (high priority). This representation ensures that as users move and generate service requests, the system can dynamically allocate resources to meet the demands efficiently while considering the priority of each service.

Let $L(t)$ denote the vector of normalized request counts for each service at time t. The $L(t)$ for a service Sv_i denoted as $L(t)_{Sv_i}$ is calculated as:

$$L(t)_{Sv_i} = \frac{\text{Number of requests for service } Sv_j \text{ at time t}}{\text{Total number of requests for all services at t}} \tag{10}$$

The placement decision is made step by step, by considering each service on a host device, within a state at time t until all available clients have been examined. The overall placement decision, represented as k, is a two-dimensional matrix of size $n \times m$. In this matrix, the value $k_{i,j}(t) = 1$ indicates that service Sv_i is deployed on device H_j at time t, while 0 signifies otherwise.

The state space is represented by $S(t) \in S$, with $S(t) = (k(t), i, j, DA(t))$, where k is the current deployment and placement matrix, i is the index of the service that we are trying to deploy, j is the current host device index, and DA is a vector of length y that indicates the requested areas for deploying clients, where DA_y is set to 1 if the area is requested and 0 if it is not. Here, y represents the total number of area locations in the environment.

The available actions in our framework are binary, offering two distinct choices: (1) selecting a host for service deployment or (2) excluding a host from the deployment process. Specifically, when a host H_j is selected, the action involves deploying service Sv_i on that host. In contrast, if a host is not chosen, the selected service remains unassigned to that device. These actions are represented as $A = [0, 1]$ where 1 indicates the deployment of the service Sv_i on the host and 0 signifies the exclusion of the host from the deployment cycle of the service.

4.2. GCT configuration

The GCT model integrates GCN and Transformer encoders, as illustrated in Fig. 1. This hybrid architecture enables the model to capture both intricate spatial dependencies and evolving temporal dynamics within the state-action graph. Specifically, the GCT framework consists of a GCN module followed by a Transformer encoder. The GCN processes the initial input of states and actions, implementing the look-ahead advice mechanism to enhance decision-making. The processed output from the GCN then serves as the input for the Transformer encoder. The final output of the GCT is a probability distribution, which represents the potential shaping function and informs resource allocation strategies.

To compute the Krylov basis, we utilize the augmented Krylov algorithm applied to a sub-graph of the system [40]. This algorithm generates a series of vectors that approximate the transition matrix's behavior. Specifically, the augmented Krylov algorithm operates by iteratively generating vectors that capture the dominant characteristics of the transition matrix based on the sub-graph. These vectors are then aggregated to form the Krylov basis, which serves as an approximation of the transition matrix. This basis facilitates the representation of the system's dynamics in a reduced-dimensional space, enabling more efficient computation.

The standard loss function for the GCT integrates both base and recursive components to accurately capture the message passing mechanism. The loss function is designed to reflect these two aspects, ensuring a comprehensive evaluation of the network's performance. Specifically, the loss function is expressed as follows:

$$loss = loss_0(\mathbf{S, A}) + Loss_{rec}(S, A) \tag{11}$$

In this formulation, the Base Loss ($loss_0$) pertains to the initial states and actions, while Recursive Loss accounts for the iterative updates based on the message passing through the network. The lists Base States and Base Actions represent the foundational data points where each action is paired with its corresponding state, and these are associated with non-zero rewards for the current episode. Conversely, S and A denote the states and actions derived from the sampled transitions, which are used to compute the recursive loss ($loss_rec$). This structure ensures that both the static and dynamic aspects of the network's operation are considered, allowing for a more accurate and effective learning process.

4.3. Reward function

Our reward shaping method is designed to simultaneously satisfy four key objective functions while avoiding specific constraints, ensuring a balanced and efficient approach to resource management. Each objective is associated with a weight value, denoted as w_i where the total weight sum $W = \sum_{i=1}^{4} w_i = 1$. These weights are fully adjustable, allowing for flexibility in prioritizing different objectives depending on the specific needs of the environment or application. The four objective functions are: (1) Maximize Service Fulfillment, (2) Minimize Host-Service Distance, (3) Maximize High-Priority Service Deployment, and (4) Minimize the Number of Deployed Hosts. By carefully adjusting the weights assigned to each objective, our method can adapt to different scenarios, striking an optimal balance between competing goals such as service coverage, resource efficiency, and priority management.

The first goal is to **Maximize Service Fulfillment** c_1. This objective is designed to maximize the number of services that are successfully deployed, ensuring the system meets user demands, even in dynamic environments where users and devices are constantly moving. The goal is to encourage the agent to strategically place services in anticipation of shifting demand at the next time-step. By learning the patterns of service demand fluctuations and user mobility over time, the agent becomes more adept at predicting and responding to these changes. In such dynamic environments, this approach ensures that a higher number of user requests are fulfilled by the edge cluster, despite the challenges posed by constantly changing locations and service requirements. This leads to lower response times, increased throughput, and overall improved QoS. The objective is to drive the agent towards maximizing service fulfillment, minimizing the cost associated with unmet demands, and ensuring efficient resource allocation that adapts to the movement of users and the evolving environment. This cost can be mathematically formulated as

$$c_1(t) = max(\sum_{i=1}^{n}\sum_{j=1}^{m} D_i(t) \times k_{i,j}(t)) \times w_1 \tag{12}$$

c_1 is summing the total number of services given that they were requested at time t, where $D(t)$ is an array of size n. Each element $D_i(t) = 1$ indicates that the service Sv_i was both requested, as denoted by $L_i(t) = 1$, and successfully deployed on a host from the set H. Each element $D_i(t)$ is defined as:

$$D_i(t) = \begin{cases} 1, & \text{if } L_i(t) = 1 \text{ and } Sv_i(t) \text{ is successfully} \\ & \text{deployed on a host from the set } H, \\ 0, & \text{otherwise.} \end{cases} \tag{13}$$

The second objective is to **Minimize Host-Service Distance** c_2. This function focuses on minimizing the physical or network distance between the hosts and the services they deploy, especially in a dynamic environment where users are constantly moving. By strategically placing services closer to where users are located or expected to be, the system reduces latency and enhances service efficiency. This approach is crucial for the fog computing layer, as it aims to deliver services as near as possible to the end-users. The on-demand architecture tracks user movements within its clusters, allowing it to compute the average time each user spends in a cluster and predict their likely next location as proved in [35]. By doing so, it aims to minimize the distance between the current service deployment host and the user's next place location, ensuring the service remains accessible as users move. This strategy helps reduce bandwidth delays and improves physical reachability, optimizing overall service performance. Prioritizing proximity not only improves response times but also ensures a higher QoS, adapting in real-time to the shifting locations and demands of users. This cost function can be expressed as:

$$c_2(t) = min(\sum_{i=1}^{n}\sum_{j=1}^{m} d_{i,j} \times k_{i,j}(t)) \times w_2 \tag{14}$$

where $d_{i,j}$ is the distance (physical or network) between host H_j and the user requesting service Sv_i, extracted from $H_{j,location}$ and the next place a user will be in a dynamic and moving environment.

The third objective is to **Maximize High-Priority Service Deployment** c_3. This objective focuses on ensuring that high-priority services are given precedence in deployment decisions, guaranteeing that essential applications receive the resources they need to operate efficiently. In a dynamic environment where users are constantly moving and generating varying demands, it's crucial to prioritize the rapid deployment of critical services. By doing so, the system maintains a high level of QoS for these vital applications, ensuring they are deployed promptly and without delay, even in the face of competing demands from lower-priority services.

$$c_3(t) = max(\sum_{i=1}^{n}\sum_{j=1}^{m} p_i \times k_{i,j}(t)) \times w_3 \tag{15}$$

where p_i is the priority of service Sv_i, derived from $Sv_{(i,priority)}$, with values ranging from 0 to 1.

The fourth objective is to **Minimize the Number of Deployed Hosts** c_4. The goal is to reduce the number of active hosts required for service deployment, which is significantly important in dynamic environments where user movements and fluctuating demands can strain system resources. By minimizing the number of hosts in use, the system conserves unused resources, simplifying management and reducing the overall complexity of the infrastructure. This approach not only lessens the burden on servers responsible for monitoring and maintaining host health but also accelerates the learning process for adapting to changing resource availability and optimizing service placement in real-time.

$$c_4(t) = min(\sum_{j=1}^{m}(\sum_{i=1}^{n} k_{i,j}(t) \geq 0)) \times w_4 \tag{16}$$

This objective reduces the number of hosts being used by ensuring services are deployed on the minimum number of hosts required.

Those objectives are subject to resource constraints. Each host H_j has limited resources $H_{i,cpu}$, $H_{i,memory}$, $H_{i,diskspace}$, and the total resources used by all services deployed on that host must not exceed the available capacity as express below:

$$\sum_{i=1}^{n} k_{i,j}(t) \times Sv_{i,cpu} \leq H_{i,cpu} \tag{17}$$

$$\sum_{i=1}^{n} k_{i,j}(t) \times Sv_{i,memory} \leq H_{i,memory} \tag{18}$$

$$\sum_{i=1}^{n} k_{i,j}(t) \times Sv_{i,diskspace} \leq H_{i,diskspace} \tag{19}$$

while a service can only be deployed on a one host at a time.

$$\sum_{j=1}^{m} k_{i,j}(t) \leq 1 \tag{20}$$

If the constraints are violated, this will affect the reward function by adding those violations as punishments to the method. The $CPU_{violation}$ can be expressed as:

$$CPU_{violation}(t) = \sum_{j=1}^{m} max(0, \sum_{i=1}^{n} k_{i,j}(t) \times Sv_{i,cpu} - H_{i,cpu}) \tag{21}$$

The same logic applies to $Memory_{violation}(t)$ and $diskspace_{violation}(t)$.

The second violation occurs if a service is deployed on more than one host simultaneously expressed as follows:

$$Deployment_{violation}(t) = \sum_{i=1}^{n} max(0, \sum_{j=1}^{m} k_{i,j}(t) - 1) \tag{22}$$

The overall reward method $r(t)$ at timestep t will be equal to $r(t) = (w_1 \times c_1(t)) - (w_2 \times c_2(t)) + (w_3 \times c_3(t)) - (w_4 \times c_4(t)) - (CPU_{violation}(t) + Memory_{violation}(t) + diskspace_{violation}(t) + Deployment_{violation}(t))$.

This optimization framework dynamically adjusts to user movements and changing demands, ensuring that the deployment of services is both efficient and responsive to the dynamic environment. The balance between maximizing service fulfillment, minimizing latency, prioritizing critical services, and optimizing host usage is achieved by tuning the weight parameters according to the system's specific needs. This reward function is later shaped by Equation 4.

4.4. Solution

Algorithm 1: PPO with GCT (GCN + Transformers) for Reward Shaping.

Input: Environment parameters, PPO parameters, GCT parameters;
Output: Trained PPO model with reward shaping;
Initialize PPO policy network π_θ;
Initialize GCT model with GCN and Transformer layers;
Initialize Rollout storage `storage`;
Initialize optimizers for π_θ and GCT models;
while *not done* **do**
  for *each* `step` *in* `num_steps` **do**
    Sample actions A_t from policy network $\pi_\theta(A_t|S_t)$;
    Step the environment, observe new states S_{t+1}, and dones d_t;
    Store experiences $(S_t, A_t, r, S_{t+1}, d_t)$ in `storage`;
    if *step* % `GCN_interval` $== 0$ **then**
      Build Transition Graph G;
      for *each transition in* `storage` **do**
        Add transition (S_t, A_t, S_{t+1}) to G;
      end
      Build the sampled transition matrix P' from G;
      Construct Krylov basis K from P';
      Update GCT with K;
    end
  end
  Compute returns r and advantages $\hat{A}_t$;
  Compute the probability ratio $pr(\theta) = \frac{\pi_\theta(A_t|S_t)}{\pi_{\theta_{old}}(A_t|S_t)}$;
  Compute the clipped objective:

$$L^{CLIP}(\theta) = \mathbb{E}_t[\min(pr(\theta)\hat{A}_t, \text{clip}(pr(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)]$$

  Update the policy parameters by maximizing the objective function:

$$\theta \leftarrow \theta + \alpha \nabla_\theta L^{CLIP}(\theta)$$

  Compute combined value function:

$$Q_{\text{comb}} = \alpha Q + (1-\alpha) Q_\phi$$

  Update rewards in `storage` with shaped rewards;
  if *episode end* **then**
    Log rewards, losses, and other metrics;
    Save model checkpoints;
  end
  Reset `storage` and environment for new episode;
end
Return: Final trained PPO policy with GCT-based reward shaping;

In dynamic environments where conditions are constantly changing, and users are frequently on the move, it is essential for RL agents to quickly and effectively adapt to these shifts. Our agent leverages the Proximal Policy Optimization (PPO) algorithm [41], a state-of-the-art policy gradient method that is particularly well-suited for environments characterized by such dynamism. The PPO algorithm optimizes the policy directly, which is crucial in scenarios where the environment is non-stationary, meaning the optimal actions can change over time as users move and as resources fluctuate.

PPO is favored in dynamic environments due to its ability to maintain stability during policy updates, which is vital when the environment is in shift. The algorithm introduces a clipped objective function, which ensures that updates to the policy do not deviate excessively from the previous policy. This stability is crucial when the agent must continuously adapt to new states and conditions without overshooting or making erratic changes that could lead to suboptimal performance. The PPO objective function is formulated as:

$$L(\theta) = \mathbb{E}_t \left[\min \left(pr(\theta)\hat{A}_t, \quad \text{clip}(pr(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right] \tag{23}$$

where θ represents the policy parameter, The clip function in PPO limits the policy update by constraining the probability ratio $pr(\theta)$ within a range around 1, and $pr(\theta)$ represents the probability ratio between the new policy and the old policy, expressed as:

$$pr(\theta) = \frac{\pi_\theta(A_t|S_t)}{\pi_{\theta_{old}}(A_t|S_t)} \tag{24}$$

Furthermore, $\hat{A}_t$ is the advantage estimate that measures how much better an action a performs compared to a baseline (usually the value function), and ϵ is a small hyperparameter that determines the range of clipping. This clipping mechanism is essential because it prevents large policy updates, which could destabilize learning in dynamic environments where the agent needs to make incremental adjustments rather than drastic changes. As the agent encounters new data, it can adjust its policy to accommodate changes in user behavior, resource availability, and other environmental factors. This adaptability ensures that the agent continues to

perform optimally even as the environment shifts, providing reliable service and maintaining high efficiency. In our context, where users are constantly moving and generating new service requests, PPO allows the agent to dynamically allocate resources in real time. By optimizing the policy to respond to changing conditions, PPO ensures that the agent can quickly adapt its strategy, effectively balancing resource usage and service quality. The process is outlined in Algorithm 1.

In parallel, we are training the GCT. At regular intervals, a subset of the transition matrix is used to construct a sub-graph, which is then employed to train the GCT. During this training, the augmented Krylov algorithm is utilized to approximate the transition matrix based on the sampled sub-graph [17]. The GCT's loss function is then applied to optimize its performance. Subsequently, the combined value function $Q_{comb} = \alpha Q + (1 - \alpha)Q_{\phi}$, which incorporates reward shaping, is derived, enabling enhanced decision-making and learning efficiency within the dynamic environment. The GCT updates the reward shaping function periodically, while the DRL agent learns to optimize based on this enriched feedback. This combined approach helps in managing complex, dynamic environments with moving users, improving overall learning efficiency and system performance. Moreover, our reward function explicitly incorporates network latency as a core optimization criterion, ensuring that the agent learns to prioritize service placements that minimize delay. As the DRL agent optimizes its policy over time, it naturally selects edge hosts with lower expected latency to users, ensuring that service allocation remains responsive even under dynamic mobility conditions. This adaptive learning approach enables our framework to achieve low-latency, high-availability service delivery without requiring direct modeling of inter-host communication overhead.

5. Experiments and analysis

In our experiments, we simulate a highly dynamic environment where users exhibit significant mobility, frequently moving between different geographic locations while requesting various critical services. An example of such services includes continuous health monitoring provided by wearable devices. To capture the intricate patterns of user movement and service requests, we utilize the MOBILE DATA CHALLENGE (MDC) dataset [42], which is notably well-suited due to its detailed records of user mobility and geographical context. This dataset allows us to model users as clients actively requesting services. The host devices that volunteer to provide these services are selected based on the Google Cluster Workload dataset [43], which offers a rich source of data on the resource capabilities and usage patterns of cloud-based systems. MDC represents a continuous record of user mobility patterns in a dynamic environment, making it ideal for analyzing user-driven service requests and resource availability fluctuations. To further validate our approach under real-world mobile edge computing conditions, we integrate the Shanghai Telecom Mobile User Dataset [44]. This dataset provides real-world cellular network mobility traces, including handover events between base stations and timestamped service requests such as data traffic, SMS usage, and call records. Unlike MDC, which primarily focuses on spatial mobility patterns, the Shanghai Telecom dataset allows us to analyze how mobile users generate service demands dynamically while transitioning between area locations. By integrating both datasets, we ensure that our evaluation considers both mobility-driven and service-driven variations, creating a comprehensive and realistic simulation of adaptive resource management in mobile edge computing environments.

To validate the effectiveness of our proposed solution, we conducted a comparative analysis involving several benchmarks. Our approach is assessed against a DRL model that does not utilize reward shaping, as well as heuristic solution [28] and reward shaping methods employing RNNs with GCN [33] (GCRN). This comparison aims to evaluate how the DRL agent performs under shifting environmental conditions, especially focusing on how it adapts to changes in movements. Additionally, we explore variations in our objective functions and observe the impact on available resources when implementing our solution.

We evaluate our approach using PPO and compare it with baseline methods. While Soft Actor-Critic (SAC) and Twin Delayed Deep Deterministic Policy Gradient (TD3) are known for their sample efficiency and policy stability, their reliance on off-policy learning and replay buffers makes them less suited for rapidly evolving environments where state transitions shift frequently. Our dynamic edge computing scenario introduces high variability in resource allocation, requiring an algorithm that can adapt to real-time changes efficiently. PPO, with its policy gradient updates and adaptive step size, achieves a better trade-off between stability and adaptability, ensuring robust performance in our use case.

The model architecture, as detailed, integrates a GCN layer with a predefined number of input features, which defines the dimensionality of the output. This output is then processed by a Transformer Encoder, which is configured with a hidden size of 32, four attention heads, two layers, and a dropout rate of 0.2 to address potential overfitting. We conducted an extensive hyperparameter tuning process using Bayesian Optimization. The key parameters tuned include the learning rate, discount factor, batch size, exploration strategy, and network architecture parameters. The tuning process aimed to balance the trade-offs between convergence speed, sample efficiency, and computational overhead in a highly dynamic environment. The hidden size of 32 was chosen to balance computational efficiency and representational power, as larger hidden sizes were found to increase training time without significant performance improvement. The selection of four attention heads was based on preliminary experiments indicating that this configuration provides an effective trade-off between model complexity and the ability to capture diverse patterns in the input. The two-layer structure of the Transformer Encoder was determined to be sufficient for the task, as additional layers did not yield noticeable gains in performance, and increasing depth could lead to overfitting in the absence of a significantly larger dataset. The dropout rate of 0.2 was tuned to mitigate overfitting, as higher values led to underfitting and lower values were insufficient for regularization. These architectural choices were informed by empirical validation and guided by considerations of computational feasibility, ensuring the model achieves robust performance while maintaining efficiency. The final output is produced by a fully connected layer that translates the transformed features into the desired output format. Our experimental setup includes 45 services and 15 devices available as hosts. We utilize the Open AI Gym [45] due to its versatility in incorporating and merging various architectures, and we developed our customized dynamic environments within this framework. In the experiments, we simulate shifts and changes in the environment

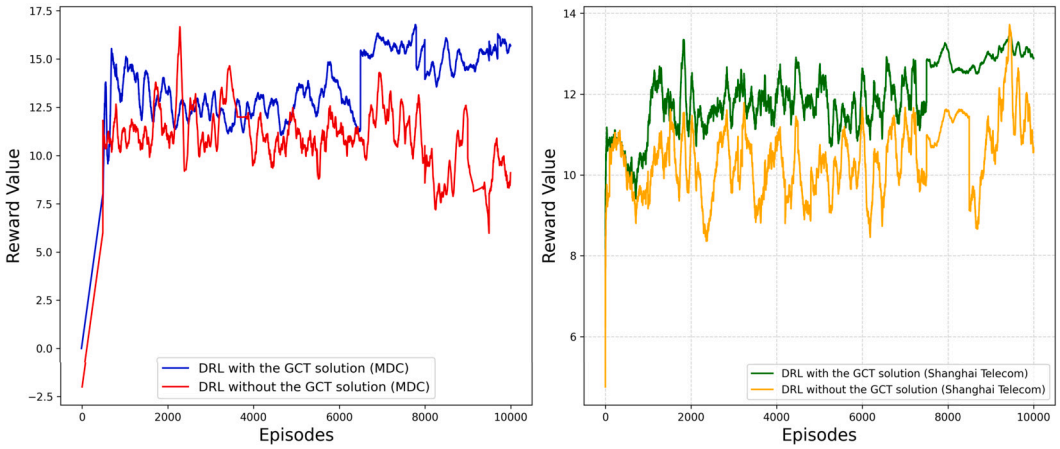

Fig. 2. The performance of a DRL agent with and without the use of the GCT solution.

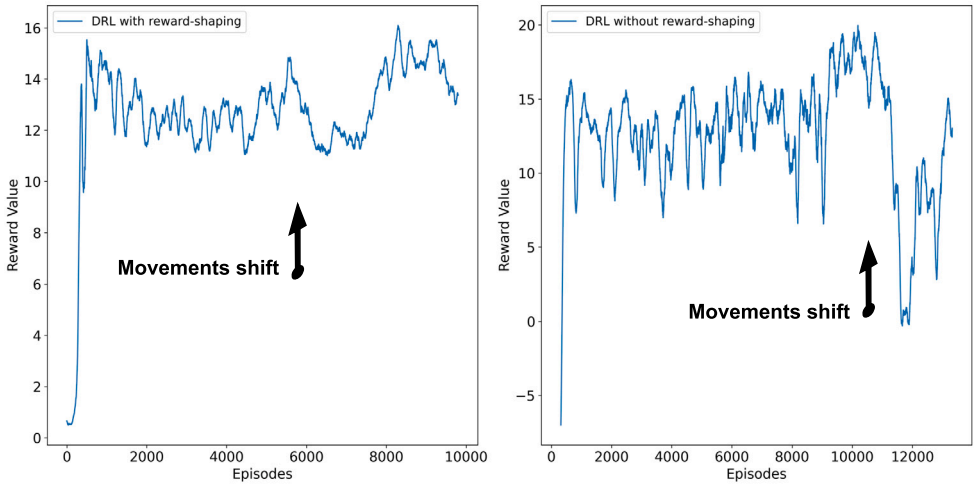

Fig. 3. The performance of our DRL agent, both with and without the implementation of reward shaping, during shifts and changes in the environment.

by dynamically altering the requests of a subset of users while the DRL agent continues to learn. Specifically, every 500 episodes, we randomly modify the requests of 5% to 10% of the users to mimic real-life scenarios where user mobility leads to changes in location, and new users arrive with different service requests. These changes directly impact the request matrix and the availability of host devices, reflecting the dynamic nature of resource management in real-world environments. Such scenarios make it crucial to have reward shaping in the dynamic environment, as it enables the DRL agent to adapt more quickly and efficiently to these changes, ensuring faster convergence and improved performance compared to standard learning approaches. This highlights the significance of reward shaping in keeping pace with evolving conditions and maintaining optimal resource allocation.

1) In our experiments, we assess the effectiveness of our proposed solution incorporating reward shaping and mobility considerations versus a baseline without reward shaping in a dynamic environment where users move frequently and request services. As depicted in Fig. 2, the DRL agent without reward shaping exhibits notable instability, struggling to achieve convergence and stable learning. This instability is reflected in the high fluctuations of reward values, particularly in the Shanghai Telecom dataset, where the reward values oscillate between 7 and 13 before stabilizing at a lower value. Similarly, in the MDC dataset, the absence of reward shaping results in fluctuations ranging between 9 and 14, with notable performance drops after 2000 episodes, indicating an inability to effectively adapt to dynamic changes.

In contrast, when incorporating reward shaping and user movement modeling, the DRL agent demonstrates improved learning stability across both datasets. In the MDC dataset, the agent achieves higher and more stable rewards, ranging from 12 to 17, with fewer fluctuations. Similarly, in the Shanghai Telecom dataset, reward shaping significantly improves adaptation, with rewards stabilizing at higher values between 11 and 14 compared to the baseline. This demonstrates that guiding the learning process through our structured reward shaping helps the agent adapt more effectively to environmental changes, resulting in faster and more stable convergence. Moreover, both datasets reached convergence around episode 8000 when reward shaping was applied. This suggests that reward shaping accelerates learning and allows the DRL agent to quickly adapt to the dynamic nature of user mobility and service demands. However, in the absence of reward shaping, the agent fails to achieve full convergence even after 10,000 episodes, with

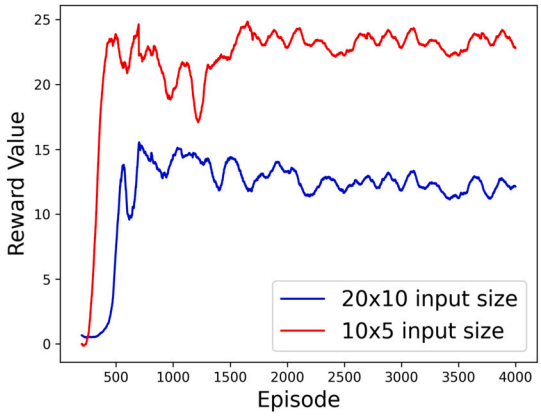

Fig. 4. The performance of our agent as we increase the scale of the environment's input with different users' movements.

reward values still fluctuating significantly. These findings highlight the critical role of reward shaping in improving DRL performance in dynamic environments, ensuring more efficient service allocation and deployment strategies.

2) To assess our model's performance under rapid changes in a dynamic environment, we introduce significant shifts in user requests and host devices, creating a challenging scenario for fast convergence. Additionally, we allow the DRL agent that is not using our solution to have more learning rounds before implementing these shifts in user movement, demonstrating that even with extended training, classical DRL methods still fail to converge quickly under dynamic conditions. The results, illustrated in Fig. 3, highlight the effectiveness of our approach. In the left graph, where reward shaping is applied, the agent demonstrates resilience to swift changes. Although there is an initial drop in reward values to approximately 5 due to the shifts, the model quickly adapts, with reward values rising to about 12.5 within the next 500 episodes. This rapid recovery indicates that reward shaping facilitates quick stabilization in response to dynamic fluctuations.

In contrast, the right graph illustrates the performance of the DRL agent without the inclusion of reward shaping. Here, the agent experiences a drop in reward values to around 2.5 and shows a slower convergence to higher values, only reaching approximately 10 after 1000 episodes. This slower adaptation underscores the critical role of guiding the agent and the importance of reward shaping in guiding the agent through rapid environmental changes. By incorporating reward shaping, our solution effectively mitigates the impact of sudden shifts and accelerates the learning process, demonstrating its superiority in managing dynamic and highly active environments.

3) To further assess the robustness of our proposed solution, we evaluated its performance across various input sizes, including configurations with 10 services and 5 hosts, 20 services and 10 hosts, as well as with 45 services and 15 hosts. As shown in Fig. 4, the model demonstrates a consistently acceptable and stable convergence rate even as the input size scales up while having different rates of user shifts and movements.

For the scenario with 10 services and 5 host devices, the reward values stabilize within the range of 22 to 25, indicating effective learning and stability despite the dynamic environment and the presence of critical services. This stability reflects the model's capability to handle varying conditions and maintain performance. Similarly, in the larger configuration with 20 services and 10 hosts, the reward values eventually settle between 12 and 15. This further illustrates that, regardless of the increased complexity, the model preserves stability and effectively adapts to larger input sizes. Moreover, as previously shown in Fig. 2, on 45 services and 15 hosts, the performance of the agent is also smooth and converges faster. The results confirm that our solution provides robust learning and consistent performance even in more complex and dynamic scenarios. This stability is achieved by incorporating the movement factor into the MDP formulation, which helps the agent adapt to user changes and mobility. By fine-tuning the reward values towards desired targets, we demonstrate the model's capability to thrive in larger and more complex environments without compromising performance. Overall, this integration of the movement factor enhances the agent's ability to respond effectively to the dynamics of user behavior.

4) Based on the analysis depicted in Fig. 5, our evaluation of resource utilization during deployment and service pushing highlights the benefits of the on-demand architecture over traditional static deployment methods. In terms of resource usage, the on-demand approach significantly improves availability, with CPU usage showing 60% for regular operations and 40% for on-demand tasks. Memory and hard disk utilization exhibit similar patterns, with on-demand usage reaching 70% and 65%, respectively, while regular usage remains around 30-35%. Battery usage is also optimized, with a 60% allocation for on-demand activities and 40% for regular usage. These figures indicate that our solution adapts dynamically to real-time demands, maximizing resource utilization while balancing power efficiency.

Furthermore, when examining the availability and usage of host devices, the on-demand deployment method uses about 30 devices, leaving around 70 devices available, whereas regular deployment requires 50 devices, with only 30 remaining available. This enhanced flexibility ensures that our approach leverages a more diverse range of host devices, thereby increasing the pool of available resources and optimizing overall usage.

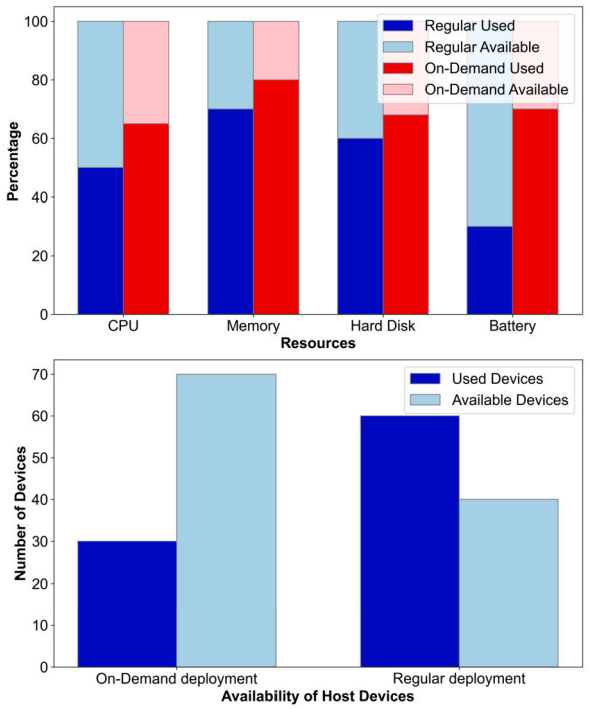

Fig. 5. The resource utilization and availability of host devices when employing on-demand deployment compared to using static host devices.

To further address computational efficiency, our analysis incorporates computational overhead, memory usage, and power consumption. The results indicate that our on-demand approach reduces the number of active devices, leading to lower overall computational costs while maintaining system responsiveness. Additionally, our architecture minimizes redundant processing, ensuring that memory utilization remains within optimal bounds while avoiding unnecessary resource exhaustion. The reduced active device count further lowers overall power consumption, making the approach more sustainable and energy-efficient. By dynamically allocating resources based on real-time demand and balancing the load effectively, our on-demand architecture supports scalable and efficient deployment, ensuring that fewer devices are actively engaged while more resources remain available to meet the dynamic needs of the environment.

5) Afterward, we experimented with modifying the weights of our objective functions, as illustrated in Fig. 6. In our reward function formulation, we assign four distinct weights to each objective function, allowing them to be adjusted based on the service provider's assessment of the environment's requirements. Altering these weights introduces corresponding changes to the rewards calculated by the agent. In the figure, a signal value of one indicates that the weight is active for the current cycle. When multiple weights are active simultaneously, the total weight is distributed equally among them.

Initially, the reward function begins to converge steadily around a value of approximately 12. However, any change in the weights triggers a peak in the reward, as depicted in the results. For instance, around iteration 4×10^6, there is a noticeable peak in the reward value, reaching up to approximately 16. This peak reflects the agent's response to the updated cost function. Each adjustment to the weights leads to a temporary spike in reward values before the agent re-stabilizes and continues to converge, demonstrating the system's sensitivity to the weight distribution and its ability to adapt to shifting priorities in real-time.

6) To evaluate the impact of our GCT-based scheduling approach on service allocation latency, we compare it against a traditional Greedy algorithm. The Greedy algorithm follows an immediate selection strategy, allocating resources based on current availability without considering long-term optimization or load balancing. While this approach is computationally lightweight, it often leads to resource contention and higher queuing delays, particularly in dynamic environments.

Our experimental results, illustrated in Fig. 7, demonstrate that our proposed GCT-based approach significantly reduces service allocation latency. Specifically, the Greedy algorithm results in an average latency of 18.5 ms, whereas our proposed solution achieves a much lower latency of 10.2 ms, representing an improvement of approximately 45%. This reduction is attributed to our method's ability to predict and optimize resource allocation decisions dynamically, preventing bottlenecks and ensuring a more efficient deployment strategy.

7) Finally, we compared our GCT approach with a GCRN that utilizes an RNN following a GCN training, as well as with heuristic methods such as Genetic Algorithms (GA) while having dynamic environments and the users are moving, as shown in Fig. 8. Unlike GCT, which incorporates Transformers for enhanced dynamic adaptation, the GCRN relies on RNNs. Additionally, heuristic approaches like GA, which depend on random processes to generate solutions, face challenges in large input scenarios due to the vast range of potential solutions, often struggling to find optimal results.

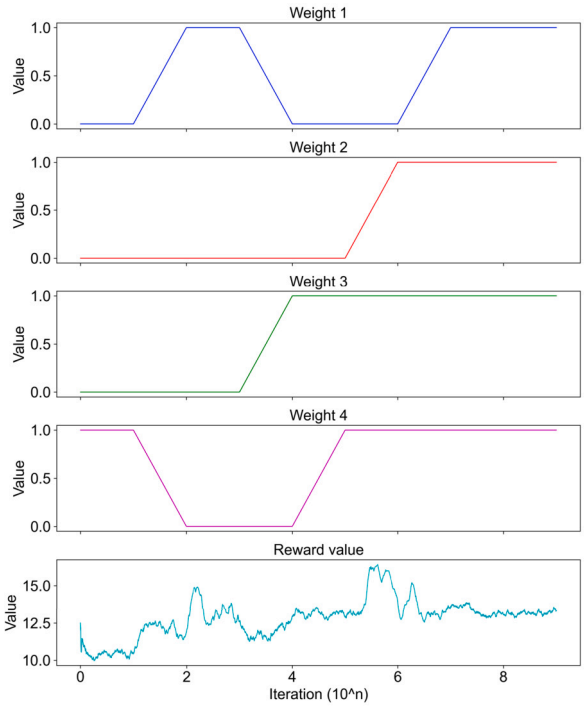

Fig. 6. GCT performance while changing weights.

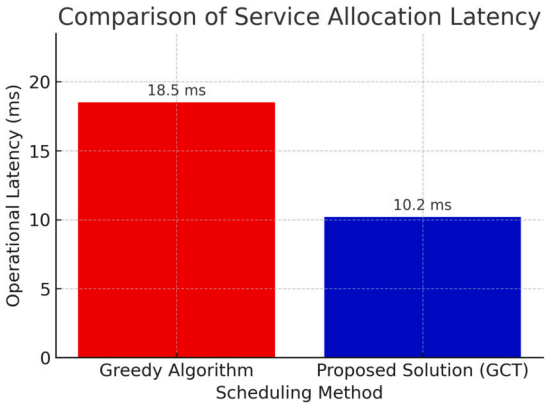

Fig. 7. Greedy vs GCT latency comparison.

In our experiments, we implemented the same cost/reward function described in Section 5 for evaluating fitness in the GA. As detailed in [35], the GA's performance often suffers from outdated placements because it periodically executes solutions that may not reflect current demands. The random nature of heuristic algorithms, combined with the increasing execution time required for larger input sizes, renders them impractical for time-sensitive and dynamic environments.

As depicted in the graph, our GCT approach consistently achieves higher reward values compared to the GCRN and heuristic methods. Specifically, GCT reaches reward values that fluctuate around an average of 14 to 16 after 6,000 episodes, peaking at values above 16 in some instances. On the other hand, GCRN shows a lower average reward, oscillating around 12 to 14. The heuristic solution performs the least effectively, maintaining reward values mostly between 9 and 11. This clearly demonstrates that GCT's ability to adapt quickly and effectively to changes in the environment provides it with a significant advantage.

In contrast, the key advantage of using Transformers over RNNs lies in their ability to handle long-range dependencies and adapt to dynamic changes more effectively. Transformers excel in capturing complex temporal and spatial relationships within the data, which is crucial for maintaining accurate and timely resource management in dynamic environments. The GCN-Transformer integration thus provides superior performance, significantly in critical applications where rapid adaptation and precise control are essential.

Our approach in general has contributed in 30% reduction on convergence time, a 25% increase in total accumulated rewards, and a 35% improvement in service allocation efficiency compared to the above-mentioned techniques.

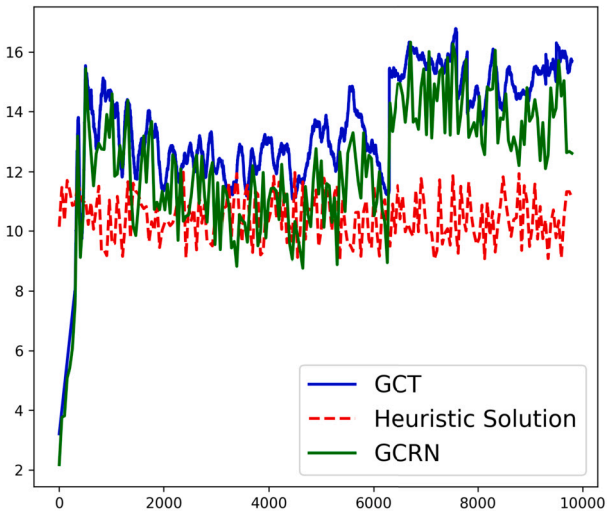

Fig. 8. The performance of GCT in comparison to GCRN and the GA heuristic approach.

6. Conclusion

In highly dynamic environments, where users are constantly on the move, building a solution that maintains high accuracy while adapting to these movements and converging quickly during such transitions poses a significant challenge. In these settings, the agent continuously learns from new data, but the evolving nature of the environment can disrupt the learning process, potentially rendering the learning rounds outdated. This complexity arises because the agent's operating conditions can shift during training, leading to an ongoing and never truly complete learning process. To address these challenges, we proposed a novel DRL framework that integrates reward shaping for the first time in the context of resource management. This framework guarantees more accurate decisions and faster adaptation to environmental changes, the proposed approach features a unique reward shaping mechanism that combines GCN and Transformer encoders, a combination not previously explored. Additionally, a novel MDP was designed to account for the dynamic nature of user mobility and demand. This integration of advanced techniques and novel MDP design significantly outperforms previous methods by providing more guided learning paths, enabling the agent to stabilize its policy more rapidly. Our approach significantly enhances the speed and accuracy of resource allocation while optimizing computational efficiency. Through our structured reward shaping, we achieved, on average across two datasets, a 30% reduction in convergence time, a 25% increase in total accumulated rewards, and a 35% improvement in service allocation compared to standard DRL and heuristic methods, mitigating the additional overhead introduced by advanced components. This enables scalable deployment in resource-constrained edge environments. We plan to incorporate additional environmental inputs, such as satellite imagery and IoT sensor data, to refine decision-making processes. Finally, we will investigate enhanced real-time optimization techniques, leveraging adaptive scheduling algorithms to further minimize computational latency in edge environments.

CRediT authorship contribution statement

Mario Chahoud: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Resources, Methodology, Investigation, Funding acquisition, Formal analysis, Data curation, Conceptualization. **Hani Sami:** Writing – review & editing, Validation, Supervision, Methodology, Investigation, Formal analysis, Conceptualization. **Rabeb Mizouni:** Writing – review & editing, Supervision, Project administration, Funding acquisition, Conceptualization. **Jamal Bentahar:** Writing – review & editing, Validation, Supervision, Project administration, Funding acquisition, Data curation, Conceptualization. **Azzam Mourad:** Writing – review & editing, Validation, Supervision, Project administration, Investigation, Funding acquisition, Formal analysis, Conceptualization. **Hadi Otrok:** Writing – review & editing, Validation, Supervision, Project administration, Funding acquisition, Formal analysis. **Chamseddine Talhi:** Supervision, Project administration, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This work is supported by the Fonds de recherche du Québec – Nature et technologies (FRQNT) and the Natural Sciences and Engineering Research Council of Canada (NSERC), Discovery Grant.

Data availability

Data will be made available on request.

References

[1] C.G. Cassandras, Automating mobility in smart cities, Annu. Rev. Control 44 (2017) 1–8.
[2] K. McMillan, K. Flood, R. Glaeser, Virtual reality, augmented reality, mixed reality, and the marine conservation movement, Aquat. Conserv. Mar. Freshw. Ecosyst. 27 (2017) 162–168.
[3] C. Yin, Z. Xiong, H. Chen, J. Wang, D. Cooper, B. David, A literature survey on smart cities, Science China. Information Sciences 58 (10) (2015) 1–18.
[4] M. Chahoud, S. Otoum, A. Mourad, On the feasibility of federated learning towards on-demand client deployment at the edge, Inf. Process. Manag. 60 (1) (2023) 103150.
[5] Z. Hu, W. Gong, W. Pedrycz, Y. Li, Deep reinforcement learning assisted co-evolutionary differential evolution for constrained optimization, Swarm Evol. Comput. 83 (2023) 101387.
[6] H. Godhrawala, R. Sridaran, A dynamic Stackelberg game based multi-objective approach for effective resource allocation in cloud computing, Int. J. Inf. Technol. 15 (2) (2023) 803–818.
[7] A.K. Sangaiah, A. Javadpour, P. Pinto, S. Rezaei, W. Zhang, Enhanced resource allocation in distributed cloud using fuzzy meta-heuristics optimization, Comput. Commun. 209 (2023) 14–25.
[8] A. Andam, J. Bentahar, M. Hedabou, Multimodal deep reinforcement learning for visual security of virtual reality applications, IEEE Internet Things J. 11 (24) (2024) 39890–39900.
[9] S. Fan, H. Liang, C.-C. Li, F. Chiclana, W. Pedrycz, Y. Dong, Optimal resources allocation to support the consensus reaching in group decision making, Inf. Fusion 110 (2024) 102451.
[10] A.K. Jirjees, A.M. Ahmed, A.A. Abdulla, J. Lu, E.M. Noori, R.N. Kareem, B.A. Hassan, H. Veisi, T.A. Rashid, Machine learning for recruitment: analyzing job-matching algorithms, Mach. Learn. 27 (2025) 1.
[11] H. Zhang, H. Wang, Y. Li, K. Long, A. Nallanathan, Drl-driven dynamic resource allocation for task-oriented semantic communication, IEEE Trans. Commun. 71 (7) (2023) 3992–4004.
[12] L. Hirsch, G. Katz, Multi-objective pruning of dense neural networks using deep reinforcement learning, Inf. Sci. 610 (2022) 381–400.
[13] G. Rjoub, J. Bentahar, O.A. Wahab, A.S. Bataineh, Deep and reinforcement learning for automated task scheduling in large-scale cloud computing systems, Concurr. Comput. Pract. Exp. 33 (23) (2021).
[14] L. Dong, N. Li, H. Yuan, G. Gong, Accelerating wargaming reinforcement learning by dynamic multi-demonstrator ensemble, Inf. Sci. 648 (2023) 119534.
[15] Y. Cao, A. Chandrasekar, T. Radhika, V. Vijayakumar, Input-to-state stability of stochastic Markovian jump genetic regulatory networks, Math. Comput. Simul. 222 (2024) 174–187.
[16] T. Radhika, A. Chandrasekar, V. Vijayakumar, Finite-time h∞ synchronization of semi-Markov jump neural networks with two delay components with stochastic sampled-data control, Bull. Sci. Math. 195 (2024) 103482.
[17] M. Klissarov, D. Precup, Reward propagation using graph convolutional networks, Adv. Neural Inf. Process. Syst. 33 (2020) 12895–12908.
[18] S.S. Kusumawardani, S.A.I. Alfarozi, Transformer encoder model for sequential prediction of student performance based on their log activities, IEEE Access 11 (2023) 18960–18971.
[19] G. Marques, C. Senna, S. Sargento, L. Carvalho, L. Pereira, R. Matos, Proactive resource management for cloud of services environments, Future Gener. Comput. Syst. 150 (2024) 90–102, https://doi.org/10.1016/j.future.2023.08.005.
[20] D. Kulkarni, M. Venkatesan, A.V. Kulkarni, Deep learning traffic prediction and resource management for 5g ran slicing, J. Inst. Eng. (India), Ser. B (2024) 1–14.
[21] W. Wu, M. Li, K. Qu, C. Zhou, X. Shen, W. Zhuang, X. Li, W. Shi, Split learning over wireless networks: parallel design and resource management, IEEE J. Sel. Areas Commun. 41 (4) (2023) 1051–1066.
[22] M. Fahimullah, S. Ahvar, M. Agarwal, M. Trocan, Machine learning-based solutions for resource management in fog computing, Multimed. Tools Appl. 83 (8) (2024) 23019–23045.
[23] D. Yang, W. Zhang, Q. Ye, C. Zhang, N. Zhang, C. Huang, H. Zhang, X. Shen, Detfed: dynamic resource scheduling for deterministic federated learning over time-sensitive networks, IEEE Trans. Mob. Comput. 23 (5) (2024) 5162–5178, https://doi.org/10.1109/TMC.2023.3303017.
[24] B. Desai, K. Patil, Reinforcement learning-based load balancing with large language models and edge intelligence for dynamic cloud environments, J. Innov. Technol. 6 (1) (2023) 1–13.
[25] H. Sami, A. Mourad, Dynamic on-demand fog formation offering on-the-fly IoT service deployment, IEEE Trans. Netw. Serv. Manag. 17 (2) (2020) 1026–1039.
[26] M. Anoushee, M. Fartash, J. Akbari Torkestani, An intelligent resource management method in sdn based fog computing using reinforcement learning, Computing 106 (4) (2024) 1051–1080.
[27] H. Wadhwa, R. Aron, Optimized task scheduling and preemption for distributed resource management in fog-assisted IoT environment, J. Supercomput. 79 (2) (2023) 2212–2250.
[28] H. Sami, H. Otrok, J. Bentahar, A. Mourad, AI-based resource provisioning of IOE services in 6g: a deep reinforcement learning approach, IEEE Trans. Netw. Serv. Manag. 18 (3) (2021) 3527–3540.
[29] J. Eschmann, Reward Function Design in Reinforcement Learning, Reinforcement Learning Algorithms: Analysis and Applications, 2021, pp. 25–33.
[30] Y. Hu, W. Wang, H. Jia, Y. Wang, Y. Chen, J. Hao, F. Wu, C. Fan, Learning to utilize shaping rewards: a new approach of reward shaping, Adv. Neural Inf. Process. Syst. 33 (2020) 15931–15941.
[31] R. Devidze, P. Kamalaruban, A. Singla, Exploration-guided reward shaping for reinforcement learning under sparse rewards, Adv. Neural Inf. Process. Syst. 35 (2022) 5829–5842.
[32] O. Marom, B. Rosman, Belief reward shaping in reinforcement learning, in: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 32, 2018.
[33] H. Sami, J. Bentahar, A. Mourad, H. Otrok, E. Damiani, Graph convolutional recurrent networks for reward shaping in reinforcement learning, Inf. Sci. 608 (2022) 63–80.
[34] D. Baburao, T. Pavankumar, C. Prabhu, Load balancing in the fog nodes using particle swarm optimization-based enhanced dynamic resource allocation method, Appl. Nanosci. 13 (2023).
[35] M. Chahoud, H. Sami, A. Mourad, S. Otoum, H. Otrok, J. Bentahar, M. Guizani, On-demand-fl: a dynamic and efficient multicriteria federated learning client deployment scheme, IEEE Internet Things J. 10 (18) (2023) 15822–15834.
[36] E. Wiewiora, G.W. Cottrell, C. Elkan, Principled methods for advising reinforcement learning agents, in: Proceedings of the 20th International Conference on Machine Learning (ICML-03), 2003, pp. 792–799.
[37] M. Toussaint, A. Storkey, Probabilistic inference for solving discrete and continuous state Markov decision processes, in: Proceedings of the 23rd International Conference on Machine Learning, 2006, pp. 945–952.
[38] M. Petrik, An analysis of Laplacian methods for value function approximation in MDPs, in: IJCAI, 2007, pp. 2574–2579.
[39] W. Yuan, K. He, D. Guan, L. Zhou, C. Li, Graph kernel based link prediction for signed social networks, Inf. Fusion 46 (2019) 1–10.

[40] C.-Q. Miao, Computing eigenpairs in augmented Krylov subspace produced by Jacobi–Davidson correction equation, J. Comput. Appl. Math. 343 (2018) 363–372.
[41] C. Yu, A. Velu, E. Vinitsky, J. Gao, Y. Wang, A. Bayen, Y. Wu, The surprising effectiveness of PPO in cooperative multi-agent games, Adv. Neural Inf. Process. Syst. 35 (2022) 24611–24624.
[42] J.K. Laurila, D. Gatica-Perez, I. Aad, B. J., O. Bornet, T.-M.-T. Do, O. Dousse, J. Eberle, M. Miettinen, The mobile data challenge: big data for mobile computing research, in: Pervasive Computing, 2012.
[43] Google, Google cluster workload traces, 2019.
[44] G. Zou, F. Zhao, S. Hu, Chestnut: a QoS dataset for mobile edge environments, arXiv e-prints, arXiv:2410.19248v1 [cs.LG], 2024.
[45] D. Cruz, J.A. Cruz, H. Lopes Cardoso, Reinforcement learning in multi-agent games: open AI gym diplomacy environment, in: Progress in Artificial Intelligence: 19th EPIA Conference on Artificial Intelligence, EPIA 2019, Vila Real, Portugal, September 3–6, 2019, Proceedings, Part I 19, Springer, 2019, pp. 49–60.