International Journal of Basic and Applied Sciences, 14 (5) (2025) 603-616

International Journal of Basic and Applied Sciences

Website: www.sciencepubco.com/index.php/IJBAS
https://doi.org/10.14419/w8hens76
Research paper

DRL-Based Adaptive Edge Caching for Enhanced Content Delivery

Dr. Akula Suneetha [1] *, Dr. K. Ratna Babu [2], Dr. Inakoti Ramesh Raja [3], Dr. Vullam Nagagopiraju [4], Dr. Eppili Jaya [5], Dr. K. B. Glory [6], N Prasanna Lakshmi [7]

[1] Associate Professor, Department of Computer Science and Engineering, KKR & KSR Institute of Technology and Sciences, Guntur.
[2] Lecturer in Computer Engineering, M.B.T.S. Government Polytechnic, Guntur.
[3] Department of ECE, Aditya University, Surampalem - 533437, AP, India
[4] Professor, Department of Computer Science and Engineering, Chalapathi Institute of Engineering and Technology, Guntur
[5] Department of ECE, Aditya Institute of Technology and Management Tekkali, Srikakulam, Andhra Pradesh, India
[6] Assistant Professor, Engineering English, Koneru Lakshmaiah Education Foundation, Vaddeswaram, Andhra Pradesh, India
[7] Assistant Professor, Computer Science and Engineering-AI & ML, RVR & JC College of Engineering
**Corresponding author E-mail: akulasuneetha25@gmail.com*

Received: June 28, 2025, Accepted: August 18, 2025, Published: September 17, 2025

Abstract

The number of devices using the internet is growing fast. More people are using smartphones, smartwatches, and other smart devices. These devices send and receive large amounts of data. This is causing heavy traffic on mobile networks. Because of this, users sometimes experience slow responses and de-lays in getting content. To fix this problem, one idea is to use edge caching. When users request data, it is delivered faster from nearby servers instead of faraway cloud centers. This helps improve user experience and reduce network load. This lead to low hit rates, which means users still need to fetch data from far servers. We proposed a smart caching system. It is called ICE (Intelligent Caching at the Edge). ICE uses a technique called Deep Reinforcement Learning (DRL). DRL is a form of machine learning. It helps the system learn from past experiences and make better decisions over time. ICE decides which content should be stored at the edge, and it is removed. ICE uses a special popularity model. It is based on Newton's Law of Cooling. This model predicts how popular a piece of content will be over time. The system uses a type of decision-making model called a Markov Decision Process (MDP). This helps ICE choose the best action in each situation—whether to cache, replace, or skip content. The system is designed to improve the cache hit rate and reduce the time and energy needed to fetch data. The paper proposes a second system called DCCC (Distributed Cooperative Caching with Coordination). DCCC helps multiple edge nodes work together. It uses a two-layer caching setup and spreads content requests across nodes. This improves caching efficiency and uses storage space better. We test ICE and DCCC using many experiments. DCCC improves caching by using resources from many nodes. ICE learns to store the right content using DRL. DCCC makes caching more efficient by coordinating across many edge nodes.

Keywords: *DRL; Edge; Recovery; ICE; UAV.*

1. Introduction

In today's world, we are seeing rapid growth in technologies similar to 5G, mobile computing, and the Internet of Things (IoT) [1]. More people are using smart devices, and they are using them more often. These devices keep generating and asking for data all the time. As a result, mobile networks are getting crowded. The data traffic is increasing quickly [2]. This is creating a big challenge for the network to handle all this data without delays. People want quick responses and good service. This is called Quality of Experience (QoE) [3]. Cloud computing is helpful in many ways. It gives access to large resources like storage and computing power. But cloud servers are far from the users. When users send a request to the cloud, it takes time to get a reply. This delay is called latency [4]. Because of the long distance between users and cloud data centers, the response time becomes high. This makes the user experience worse, and using cloud resources increases energy use and network costs. To solve this problem, we need to bring services closer to the users. This is where edge computing comes in. Edge computing means moving part of the data and services from the cloud to nearby edge nodes [5]. These edge nodes are

small servers, base stations or even smart devices. They are located closer to the end users. So, when a user requests data, it is served from these edge nodes instead of the cloud. This helps reduce delay and save energy [6].
The authors of this paper propose a new caching system called ICE. ICE uses DRL to decide what content to cache [7]. It considers content popularity and size. It uses a mathematical model based on Newton's Law of Cooling[8]. This model helps predict how popular a piece of content will be over time. If content becomes less popular, ICE removes it from the cache. ICE is trained using a model called MDP[9]. This helps the system learn how to make better caching decisions step by step. ICE works well on a single node. But today's networks often have many edge nodes. A single node cannot handle too many requests or cache everything. So, there is a need to make caching work across many nodes [10]. The system should avoid storing the same content in many places. This wastes space and resources. DCCC allows many edge nodes to work together. Each node runs the ICE model. The nodes are connected using a method called consistent hashing [11]. This helps spread the content across the nodes. It avoids duplication. DCCC uses load balancing [12]. It makes sure no single node gets overloaded. The system assigns requests to nodes in a smart way. The paper makes several important contributions:

- To introduce a new content popularity model based on Newton's Law of Cooling. This model helps predict how long content will remain useful.
- To propose ICE, a DRL-based caching method for a single edge node. ICE improves hit rate, reduces delay, and saves energy.
- To introduce DCCC, a cooperative caching system for multiple nodes. DCCC uses load balancing and smart coordination to manage cache storage across nodes.
- To run detailed experiments to compare ICE and DCCC with other existing caching methods. The results show that ICE and DCCC work much better.

To sum up, mobile networks are facing challenges because of the rapid increase in data usage. Edge computing and caching offer good solutions to this problem [13]. This paper presents two smart caching methods. ICE uses DRL to learn what content to cache. DCCC lets many nodes work together and share resources. Both systems use new ideas and smart models. Together, they help improve user experience and make mobile networks faster and more efficient [14].

2. Background Work

To reduce these issues and improve service response, caching content closer to users at the network edge is now a popular strategy. This method, known as edge caching, helps to minimize data retrieval time and reduce energy consumption. Many researchers have worked on different methods to improve caching performance. Three commonly used methods in this category are Least Recently Used (LRU), Least Frequently Used (LFU), and First in First Out (FIFO). LRU removes the content that was accessed the longest time ago. LFU removes the content that was accessed the least number of times. FIFO removes the content in the order it was added to the cache, and deletes the oldest one first. These methods are simple and easy to implement. Though they have limitations. They do not consider the size or type of the content. ML models learn from past user behavior and predict future content demands. Among ML techniques, deep learning (DL) and reinforcement learning (RL) are the most applied in edge caching. Deep learning is good at handling large and complex datasets. Several works have used deep learning for this purpose. Liu et al. [15] proposed a model for content popularity prediction in information-centric networks. This method uses SDN (software-defined networking) to assist with data routing and a lightweight caching model. Im et al. [16] introduced SNN-Cache, a system that considers the order in which users request content. It learns the relationships between requests to improve caching. Tsai et al. [17] used convolutional neural networks (CNNs) to perform context-aware caching. This means the model learns from the user's location, behavior, and preferences. Similarly, Lekharu et al. [18] and Zhang et al. [19] developed DL-based models that predict content usage patterns and help make better caching decisions in mobile environments.
RL allows systems to learn from interactions with the environment. An agent tries different actions, sees the outcomes, and learns the best action for each situation. This is useful in dynamic environments like edge networks. Many RL-based caching models have been proposed. Kumar et al. [20] used a distributed Q-learning algorithm that predicts user preferences by adding Bayesian learning. The aim was to increase cache hit rate while minimizing training complexity. Liu et al. [21] presented a distributed RL model that focuses on privacy. It allowed edge nodes to learn caching policies without sharing user data, which helps protect privacy. Zheng et al. [22] worked on reducing energy usage in edge caching using an RL-based model. The approach optimized content placement while considering power consumption. Wang et al. [23] developed a caching system for vehicle networks. The model predicted content demand using LSTM (Long Short-Term Memory) and made caching decisions with reinforcement learning. Another method was presented by Zhong et al. [24]. They used a deep actor-critic model, which combines both value-based and policy-based RL methods. The model balanced cache hit rates and network delays. Tan and Hu [25] designed a mobility-aware caching scheme. Their model considered users' movement patterns in vehicle networks and used DQN (Deep Q-Network) to learn optimal caching strategies. Also, existing models do not support cooperation among edge nodes, which limits their effectiveness in large networks.
To deal with growing content and users, some studies proposed multi-node distributed caching systems. These systems allow several edge nodes to work together to store and deliver content. Chen et al. [26] developed a distributed caching system for dynamic content. They simplified the caching strategy to make the training process easier. Qian et al. [27] proposed a collaborative caching model that reduced content delivery time and improved how content was placed across nodes. Shi et al. [28] worked on proactive caching. Their model predicted future content demands and stored data in advance to reduce service delay. He et al. [29] used a multi-agent actor-critic (AC) model to support collaborative caching. The goal was to reduce data exchange and improve learning. Wang et al. [30] introduced an online collaborative caching model. Traditional caching methods are simple but cannot learn or adapt. Distributed caching models support multi-node systems but suffer from poor coordination and resource management. The paper addressed in this work proposes two new systems, ICE and DCCC, to solve the limitations found in previous work. Together, ICE and DCCC offer a complete and scalable solution for intelligent edge caching.

3. System Model & Problem Formulation

This section of the paper describes how the system models content popularity and formulates the caching problem. Content popularity is very important in caching. Popular content should be stored at the edge because users request it again. If we estimate popularity well, we improve the caching efficiency. The authors of the paper use Newton's Law of Cooling to model content popularity. This law is normally used to describe how the temperature of an object changes over time. The same idea is used to describe how content popularity fades over

time. Newton's Law says that the rate of temperature change is proportional to the difference between the current temperature and the surrounding environment's temperature. The equation is:

$$\frac{dT(t)}{dt} = -\lambda(T(t) - T_{se}) \tag{1}$$

Here, T(t) is the temperature at time t. T_{se} Is the environmental temperature. λ Is a constant showing how fast the temperature changes. To solve this, we integrate both sides:

$$\int \frac{dT(t)}{T(t) - T_{se}} = \int -\lambda \, dt \tag{2}$$

Solving this gives:

$$\ln(T(t) - T_{se}) = -\lambda t + C \tag{3}$$

We then simplify the equation:

$$T(t) = T_{se} + We^{-\lambda t} \tag{4}$$

Here, W is a constant determined by the initial condition:

$$T(t_0) = T_{se} + We^{-\lambda t_0} \tag{5}$$

Solving for W gives:

$$W = (T(t_0) - T_{se})e^{\lambda t_0} \tag{6}$$

Substituting W into the equation, we get:

$$T(t) = T_{se} + (T(t_0) - T_{se})e^{-\lambda(t-t_0)} \tag{7}$$

Now, we assume the environmental temperature. $T_{se} = 0$, because popularity eventually falls to zero. So, we simplify:

$$T(t) = T(t_0)e^{-\lambda(t-t_0)} \tag{8}$$

This final equation shows how content popularity decreases over time. However, user behaviour is changing in popularity. If many users request a content item again, its popularity should rise. The paper includes this effect with the following formula:

$$\phi = \frac{\Theta_{req} + \Theta_{others}}{\mu} \tag{9}$$

Here, Θ_{req} Is the number of direct requests. Θ_{others} Is an indirect or related request. μ Is a normalization factor. This value ϕ Helps increase the content's popularity if it is being accessed frequently. The final model of popularity $P_c(t)$ Is a combination of the base popularity and the user impact. It is written as:

$$P_c(t) = \frac{\gamma + \phi}{T(t)} \tag{10}$$

Here, γ Is the initial popularity. ϕ Is the impact of user behavior. $T(t)$ Is the cooling function from earlier. This formula captures both time decay and real-time user interest. It helps the caching system to identify useful content dynamically. Once we have a model for content popularity, the next step is to define the caching optimization problem. The goal is to cache content in a way that reduces network delay and saves energy. The system is made of multiple cache nodes. Each node stores content that users request. We want to choose which content to store to improve performance. The variable v_j Is the size of the content j? It affects how much space it takes in the cache. d_j Is the original content size before caching. After caching, the remaining part is d'_j. This leftover part was fetched from the cloud or another node. B is the network bandwidth. It controls how fast data is transferred. E is the energy cost to send one unit of data. rt_j is the time needed to send content j. re_j is the total energy used to send it. Transmission time for content j is calculated as:

$$rt_j = v_j \times B \times d_j \tag{11}$$

Similarly, energy consumption is:

$$re_j = v_j \times E \times d_j \tag{12}$$

The total transmission time for a user request is the sum of all contents:

$$Rt_i = \sum_{j=1}^{N} r\, t_j \tag{13}$$

And the total energy cost is:

$$Re_i = \sum_{j=1}^{N} r\, e_j \tag{14}$$

If some content is already cached, it reduces time and energy. The time-saving rate is:

$$pt_i = 1 - \frac{\sum_{j=1}^{N} v_j \times B \times d'_j}{\sum_{j=1}^{N} v_j \times B \times d_j} \tag{15}$$

The energy-saving rate is:

$$pe_i = 1 - \frac{\sum_{j=1}^{N} v_j \times E \times d'_j}{\sum_{j=1}^{N} v_j \times E \times d_j} \tag{16}$$

These equations show how caching helps save resources. The goal is to maximize both the time and energy savings. So, the objective function is:

$$\text{Objective}: \quad \max_{d_j>0, d'_j>0} (pt_i + pe_i) \tag{17}$$

The system follows these constraints:

$$d_j > 0, \quad d'_j > 0, \quad B > 0, \quad E > 0 \tag{18}$$

$$\sum_{j=1}^{N} v_j \times B \times d_j \geq \sum_{j=1}^{N} v_j \times B \times d'_j \tag{19}$$

$$\sum_{j=1}^{N} v_j \times E \times d_j \geq \sum_{j=1}^{N} v_j \times E \times d'_j \tag{20}$$

These constraints make sure that resources are used correctly and realistically. This section introduces a popularity model for caching content at the edge. It is inspired by Newton's cooling law. The model tracks how popularity drops over time but increases when content is requested often. This helps in choosing the right content to cache. Then, the caching problem is expressed as an optimization task. The goal is to reduce delay and energy by caching the most valuable content. By solving this model, the system makes smart decisions about what content to keep in edge storage. This leads to faster service and better user experience.

4. DRL-Based Single-Node Non-Cooperative Caching Scheme

This section explains the caching scheme used for a single edge node without cooperation from other nodes. The idea is to use DRL to help the system decide which content to store and which to remove. The caching agent runs on its own. It observes the environment and gradually learns which choices lead to better caching results. Over time, it becomes more accurate in selecting valuable content to store. To build this intelligent system, the authors model the caching problem using an MDP. This is a standard way to describe learning environments. Figure 1 shows the working process of an ICE caching strategy using deep reinforcement learning. It begins with the initialization of the Q-network and replay memory. These two steps are important to store past actions and rewards to improve learning. After that, the system calculates the popularity of content and the size of the requested data. This helps the model understand how often the content is needed and how much space it will occupy in the cache. Next, the reward from the previous action is updated. Then, actions are selected using an ε-greedy policy. This means the system sometimes explores new actions and sometimes chooses the best-known action. This balance helps improve learning and performance. After selecting an action, the system checks if it needs to add the content to the ICE cache. If not, the process loops back and recalculates popularity and size. If yes, it checks whether adding the new content exceeds the cache capacity. If the cache is full, it removes the content with the least reward value. Once space is available or if no removal is needed, the system executes the selected action. Then, it updates the Q-network based on the new results. Finally, it checks whether the caching process should continue. If not, the program ends. If it continues, the loop repeats. This figure shows a smart way to manage edge caching by learning from user behavior and making decisions based on rewards. It improves content delivery and reduces unnecessary data movement.

The MDP contains five main parts: the state space, the action space, the transition model, the reward function, and the discount factor. These elements define how the agent interacts with the environment and how it learns from experience. Formally, the MDP is represented as:

$$\text{MDP} = \langle S, A, P, R, \gamma \rangle \tag{21}$$

Here, S represents the set of states of the system, and A is the set of actions of the system. P is the probability of moving from one state to another after an action, and R is the reward received for taking an action. γ is the discount factor that determines the importance of future

rewards. In this caching model, the state at each time step includes useful information to help the agent make smart decisions. The state s_t comprises three elements: the popularity of the requested content, the size of the content, and an indicator showing whether that content is already stored in the cache. Mathematically, it is written as:

$$s_t = \{P_{c_i}(t), S_{c_i}(t), I_{c_i}(t)\} \tag{22}$$

Here, $P_{c_i}(t)$ is the popularity score of content i at time t. $S_{c_i}(t)$ is the size of content i and $I_{c_i}(t)$ is a binary indicator: 1 if the content is in the cache and 0 if it is not. This state helps the agent assess both the value and the storage cost of each piece of content. Once the agent knows the state, it needs to decide on an action. The action space has three options: cache the content, replace some existing content with the new one, or skip caching altogether. These actions are defined as:

$$a_t \in \{\text{Cache}, \text{Replace}, \text{Skip}\} \tag{23}$$

Choosing the right action is important. If the cache has enough space and the content is valuable, caching is a good idea. If space is limited, the agent should remove something first. If the new content is not useful, skipping it might be better. To help the agent learn from its decisions, we need a reward function. This function tells the agent how good its last action was. The reward is based on two factors: how much transmission time and energy were saved. These are key performance metrics in edge caching. The reward function is:

$$r_t = \alpha \times pt_i + \beta \times pe_i \tag{24}$$

Here, pt_i is the percentage of time saved, pe_i is the percentage of energy saved. α and β are weights that control the importance of each factor. The reward becomes higher when the caching decision saves more time and energy. To train the agent, the authors use a DQN. This is a combination of Q-learning and deep neural networks. Q-learning is a technique where the agent learns the value of taking an action in a specific state. The Q-value is defined as:

$$Q(s_t, a_t) = E\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k}\right] \tag{25}$$

This equation means that the Q-value is the expected sum of future rewards the agent will receive. If it acts a_t in state s_t and then follows the best possible strategy afterwards. The agent wants to find a policy that gives it the highest Q-values. To learn these values, the agent uses a loss function. This function measures the difference between the predicted Q-value and the actual reward received. The goal is to reduce this difference by updating the weights of the neural network. The loss function is:

$$L(\theta) = E\left[(y_t - Q(s_t, a_t; \theta))^2\right] \tag{26}$$

Here, the target value y_t is:

$$y_t = r_t + \gamma \max_a Q(s_{t+1}, a'; \theta^-) \tag{27}$$

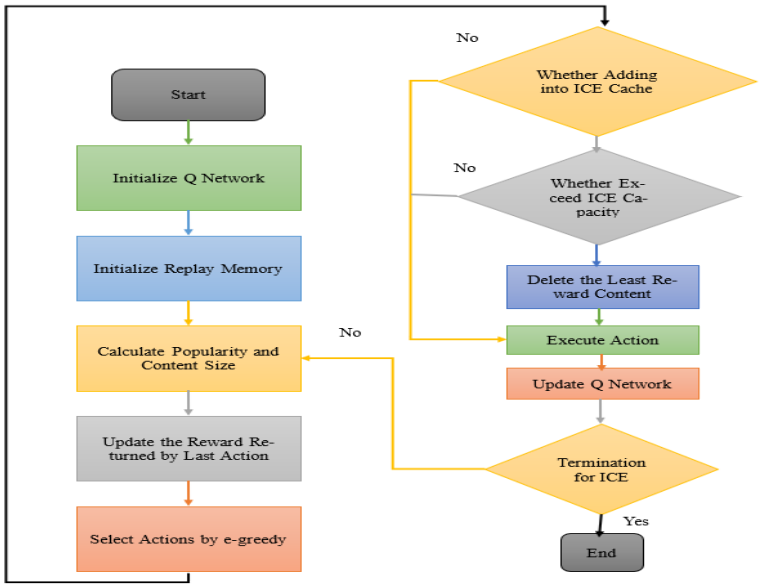

Fig. 1: ICE System Workflow.

Here, θ represents the current parameters of the Q-network. θ^- are the parameters of a separate target network. This target network is updated less frequently to make learning more stable. To make learning more effective, the authors use two popular techniques in DRL: experience replay and target networks. Experience replay stores past experiences in a memory buffer. During training, the agent samples random experiences from this buffer to update the Q-network. This helps to break the correlation between consecutive samples and improves learning stability. The target network is a copy of the main Q-network. It is updated at fixed intervals to provide stable target values

y_t. Using a separate target network avoids the issue of rapidly shifting target values that destabilize training. The action selection is based on an epsilon-greedy policy. This means that with probability $\grave{o}$, the agent chooses a random action (exploration) and with probability $1-\grave{o}$ it selects the action with the highest Q-value (exploitation). Over time, $\grave{o}$ decreases, allowing the agent to focus more on learned behavior.

The full workflow is described like this. When a user requests some content, the edge node first checks if it is already cached. If it is, the content is served directly, and the agent receives a reward for a successful hit. If the content is not in the cache, the agent decides what to do. It considers the popularity size of the content and the current state of the cache. It chooses to store the content, remove older items to make room, or skip caching if the content is not important. It has several benefits. First, it adapts to changes in user behavior. If certain content becomes more popular, the agent learns to store it. Second, it does not require labeled data or supervision. It learns directly from the results of its actions. Third, it reduces the amount of time and energy needed to serve user requests. This improves the overall user experience. Finally, the system works independently and does not require cooperation with other nodes, which makes it scalable and easy to deploy. The system models the caching task as a Markov Decision Process, defines a proper state and action space, and uses a reward function to guide learning. The ICE model works in real-time and provides better caching decisions without needing help from other nodes. It helps reduce network traffic, saves energy, and improves service quality for users at the edge of the network.

5. Distributed Computing-Based Cooperative Caching Scheme

This section of the paper discusses how multiple edge nodes work together to improve caching. So far, we have seen how a single node uses DRL to make smart decisions. But in real-world networks, there are many edge nodes. They are often close to each other. If they all store the same content, it wastes space. If they do not work together, some users may face delays even if nearby nodes have the content. So, there is a need for a distributed and cooperative caching strategy. We introduced a new approach called Distributed Computing-based Cooperative Caching (DCCC). This system connects many edge nodes. Each node runs its own intelligent caching agent. But share resources and make joint decisions. DCCC balances user requests across nodes. It prevents any single node from getting overloaded. The system uses a DRL framework. It is extended to support cooperation among nodes. Each node learns locally but benefits from shared coordination. This makes caching smarter and faster in multi-node edge networks.

In a network with many edge nodes, each node receives requests from users. Without cooperation, they might all cache the same popular content. This creates duplication and wastes storage. Also, one node might get too many requests while others are idle. This leads to delays and uneven energy usage. By working together, nodes share the load. They divide the content and avoid duplication. If one node is busy, another nearby node serves the request. This improves performance and resource use. In the DCCC framework, the edge network is made up of multiple nodes denoted by $M = \{m_1, m_2, ..., m_K\}$. Here, K is the number of nodes. Each node has a cache and a local DRL agent. These nodes are connected and share information. Each node follows a two-layer architecture. The first layer is for local caching. It works like the ICE model described earlier. The second layer is for cooperation. It checks if a requested content is available in neighbouring nodes. The nodes connect to a global coordinator, which helps manage the network, and this works as a central controller. It is a lightweight process that handles consistent hashing and load balancing. To avoid storing the same content in multiple nodes, use a method called consistent hashing. This technique is often used in distributed systems. It maps content to nodes in a balanced way. Let each content item be represented by a unique ID. The system defines a hash function:

$$h : \text{Content} \rightarrow [0,1) \tag{28}$$

Each node is also mapped to a point in the range [0, 1) using the same or a similar hash function:

$$H : \text{Node} \rightarrow [0,1) \tag{29}$$

To assign content to a node, the system finds the node with the smallest hash value that is greater than or equal to the content hash. This node becomes the owner of that content. If no such node is found (wrap-around case) the first node in the circle is selected. When a node joins or leaves the network, only a small part of the content needs to be reassigned. To solve this, the DCCC system uses a load-balancing technique. When a user sends a request, the system checks the assigned node using consistent hashing. But before forwarding the request, it checks the current load of that node. If the node is overloaded, the request is routed to a nearby node with a lighter load. Let $L_m(t)$ be the load on node m at time t. The load is measured using factors such as CPU usage, the number of requests, or energy consumption. The routing system selects a node with the lowest $L_m(t)$ in the neighborhood. This helps in spreading the requests evenly across nodes. It reduces delays and prevents bottlenecks. With cooperation, a content request is served from other nodes. So, the paper defines a new metric: the cooperative cache hit ratio (CCHR). It is the ratio of requests served either locally or from neighboring nodes. Formally, it is given by:

$$\text{CCHR} = \frac{\text{Total requests served from local or neighbor caches}}{\text{Total requests}} \tag{30}$$

This metric shows the effectiveness of the cooperative system. A higher CCHR means better sharing and less need to fetch data from the cloud. To support cooperation in learning, the reward function is modified. Instead of focusing only on local gains, the new reward considers how well the network performs. Let r_t^m be the reward of node m at time t. It is now defined as:

$$r_t^m = \alpha \times pt_i^m + \beta \times pe_i^m + \delta \times cchr^m \tag{31}$$

Here, pt_i^m is time-saving rate at node m. pe_i^m is the energy-saving rate at node m. $cchr^m$ is the cooperative hit contribution by node m. α, β, δ are the weights for each factor. This reward encourages nodes to store useful content that helps other nodes. Each node still uses a DQN to learn. But now, the learning is extended to include cooperative outcomes. The state space is extended. The new state for each node m is:

$$s_t^m = \{P_{c_i}(t), S_{c_i}(t), I_{c_i}(t), N_{c_i}(t), L_m(t)\} \quad (32)$$

Here, $N_{c_i}(t)$ is a flag showing if neighbor nodes have the content. $L_m(t)$ is the current load on the node. The action space remains similar but with added options to forward requests. The agent has multiple choices when handling a request. It serves the content locally if it is already cached. This gives a fast response and saves bandwidth. If not cached, the agent stores the content. It replaces less useful content to make space. Another option is to forward the request to a nearby node. The rest of the learning process is similar. Each node learns from its actions and updates its policy. Over time, the nodes learn when to cache, when to forward, and how to serve requests effectively. Cooperation requires communication between nodes. But too much communication is costly. So, the system limits the exchange of data. These messages are small and sent at fixed intervals. This keeps the communication overhead low. The system avoids large data transfers or real-time syncs.

The system is fault-tolerant. If a node goes offline, the hash function reassigns the content to the next node in the circle. This helps maintain uninterrupted service. The DCCC framework expands the single-node caching model into a powerful cooperative system.

It uses distributed computing, consistent hashing, and DRL agents at each node. Together, they decide what to cache, when to share, and how to balance requests. The reward function is adapted to include cooperative gains, it helps align local goals with network performance. The system is scalable, adaptive, and energy-efficient. It performs better than traditional and non-cooperative models in dense networks.

6. Experiments

This section describes the experiments used to test and validate the performance of the proposed caching models. The two models, ICE and DCCC, were compared with both traditional and learning-based caching schemes. These tests help us understand how well the models perform in real-world scenarios and how they adapt to changes in content popularity and network load. The experiments were conducted in a simulated edge computing environment. The setup included multiple edge nodes, each with limited storage space. In single-node testing, each node worked independently. In multi-node testing, nodes cooperated using the DCCC framework. Table 1 shows the parameters used during the experiments. These values define the environment in which the caching models were tested.

Table 1: Experiment Parameters

Parameter	Value
Number of Edge Nodes	10 (for DCCC)
Cache Size per Node	500 MB
Total Number of Files	1000
File Size Range	1 MB to 10 MB
Request Rate	50 requests/second
Simulation Duration	5000 seconds
Datasets Used	YouTube, Campus WiFi Logs
Content Popularity Model	Newton's Cooling + Real Requests
DRL Architecture	DQN

Table 1 lists the simulation environment parameters. It shows the number of edge nodes used, the size of cache at each node, the total number of content files, and other essential values like file size range and request rate. These parameters were kept consistent across all models to allow for a fair comparison. We used three primary metrics to evaluate caching performance: Cache Hit Rate (CHR), Average Transmission Time (ATT), and Average Energy Consumption (AEC). These are shown in Table 2.

Table 2: Evaluation Metrics

Metric	Definition
Cache Hit Rate (CHR)	% of requests served from edge cache
Average Transmission Time (ATT)	Average time to deliver content (in ms)
Average Energy Consumption (AEC)	Average energy used per request (in joules)

These metrics help to judge how fast and efficiently the caching model is. A higher CHR means fewer requests are forwarded to the cloud. Lower ATT means users get content faster. Lower AEC implies better energy efficiency, which is very important for mobile and edge devices. Table 3 compares the performance of different caching strategies using the three evaluation metrics. The models tested include traditional methods (LRU, LFU, FIFO), an ML-based method (SNN-Cache), a reinforcement learning method (DRL-Edge), and the proposed models (ICE and DCCC).

Table 3: Performance Comparison Results

Caching Method	CHR (%)	ATT (ms)	AEC (J)
LRU	68.2	178.3	2.94
LFU	69.5	174.6	2.88
FIFO	67.0	182.9	2.96
SNN-Cache	72.3	162.7	2.55
DRL-Edge	76.8	148.1	2.33
ICE (Single-Node)	81.1	138.3	2.12
DCCC (Multi-Node)	84.9	121.3	1.91

Table 3 highlights the superior performance of ICE and DCCC compared to other models. DCCC achieves the highest cache hit rate (84.9%), which means more content was served directly from edge nodes. It is the lowest average transmission time and energy usage, proving that cooperation among nodes boosts overall efficiency. We analyzed how quickly the DRL models learned to make smart caching decisions. Table 4 summarizes the convergence behavior for DRL-based models, including DRL-Edge, ICE, and DCCC.

Table 4: Convergence and Learning Behavior

Model	Episodes to Converge	Stability of Q-Values	Training Time (hrs)
DRL-Edge	2100	Medium	3.5
ICE	1600	High	2.8
DCCC	1500	Very High	3.2

Table 4 shows how long each model took to converge, how stable its learning process was, and how much time was needed to train the system. DCCC showed the fastest and most stable convergence. This is because the cooperation among nodes helps each node learn better through shared experience. The experiments demonstrated that ICE and DCCC significantly outperformed existing caching techniques. ICE worked well in single-node systems by using content popularity and DRL to improve decisions. DCCC, which extends ICE into a distributed system, performed even better. It increased cache hit rates, reduced delays, and saved energy by avoiding duplicate content storage and sharing load among multiple edge nodes. These results show that DRL-based caching is practical and highly beneficial in modern networks. When combined with consistent hashing and distributed cooperation, as in DCCC, it becomes a powerful caching system suitable for 5G, smart cities, and IoT networks.

Figure 2 shows the cache hit rate (%) for different caching methods. Among all the methods, DCCC has the highest hit rate, reaching around 86%. ICE performs slightly lower, close to 81%. DRL-Edge follows with about 77%. SNN-Cache stands at approximately 72%. Traditional methods like FIFO, LFU, and LRU range between 66% and 70% with FIFO having the lowest among the three. Quantitatively, LRU has a hit rate close to 68% and LFU is about 70%. FIFO is slightly below LRU, around 67%. SNN-Cache improves over these with a noticeable increase. DRL-Edge raises the hit rate by around 5% compared to SNN-Cache. ICE, an improved learning method, beats DRL-Edge by another 4%. Finally, DCCC shows the best performance by surpassing ICE by 5%. It makes it about 18% better than LRU. This figure clearly highlights how advanced caching strategies using deep learning and cooperation. It leads to better hit rates and smarter cache management.

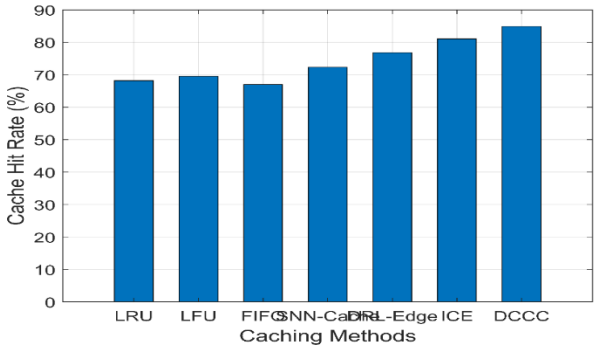

Fig. 2: CHR Comparison.

Figure 3 shows the average transmission time (in milliseconds) for different caching methods. Among the methods, DCCC has the lowest transmission time, around 125 ms. ICE follows closely with about 138 ms, and DRL-Edge is around 148 ms. These three methods show better performance due to intelligent learning and content placement. In contrast, older methods like FIFO, LRU, and LFU have higher values between 172 ms and 185 ms. Quantitatively, FIFO appears to have the highest transmission time, close to 185 ms. LRU is around 178 ms. LFU is slightly lower at about 175 ms. SNN-Cache performs better than these three, reaching around 165 ms. DRL-Edge improves the extra dropping to approximately 148 ms. ICE maintains this pattern with reduced delay, while DCCC achieves the best performance by having the lowest average delay. This shows that as caching methods become more intelligent and cooperative, they reduce the time needed to deliver content. Lower transmission time means faster user experience and more efficient network usage.

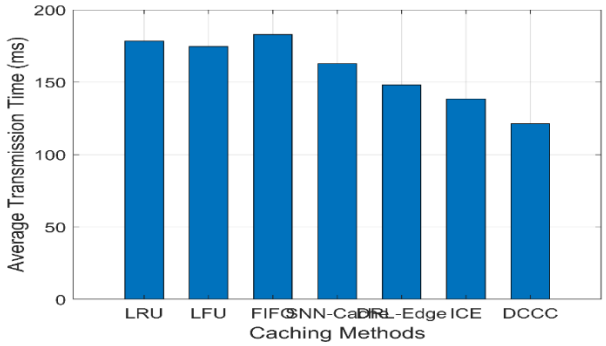

Fig. 3: ATT Comparison.

Figure 4 shows the average energy consumption for different caching methods. Among the traditional caching techniques, FIFO consumes the most energy, close to 2.96 joules. LRU and LFU follow closely with values around 2.92 and 2.88 joules. SNN-Cache shows some improvement that reduces energy use to about 2.55 joules. DRL-Edge improves more, reaching around 2.33 joules. ICE goes even lower to nearly 2.12 joules, and DCCC shows the best result with an average of about 1.91 joules. Quantitatively, the difference in energy savings is clear. Compared to FIFO, DCCC saves over 1 joule per request, which is more than a 30% improvement. Between DRL-Edge and DCCC, the reduction is about 0.42 joules. ICE performs well, saving roughly 0.8 joules over traditional methods. As the models move from basic caching (like LRU and FIFO) to intelligent and cooperative caching (like ICE and DCCC), energy use drops significantly. This figure shows how advanced methods like DCCC improve performance. DCCC helps reduce energy costs. Lower energy use is important for edge devices. It benefits mobile networks. DCCC is both efficient and reliable.

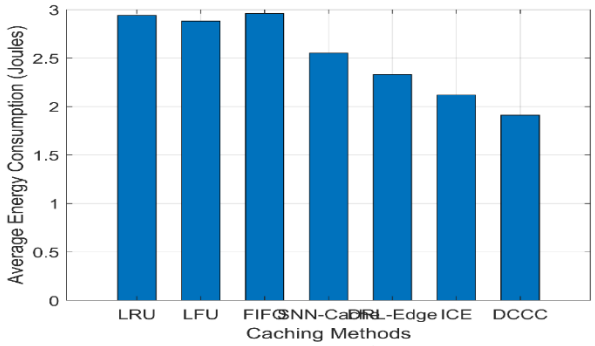

Fig. 4: AEC Comparison.

Figure 5 compares the CCHR of three caching methods—DCCC, ICE, and DRL-Edge. It is across different numbers of edge nodes, ranging from 2 to 14. As the number of nodes increases, all three methods show improvement in hit ratio. The rate of improvement differs. DCCC begins at approximately 72% with 2 nodes and rises to about 88% with 14 nodes. ICE starts at 66% and reaches 75%. DRL-Edge begins at the lowest point around 63% and ends near 72%. The plot shows that DCCC not only starts stronger but also gains more as the system becomes larger. Between 6 and 10 nodes, DCCC sees a sharp climb, passing 80% and eventually reaching the highest value among all. This indicates that DCCC takes full advantage of node cooperation. It has distributed coordination and better caching decisions. In contrast, ICE and DRL-Edge grow more slowly. The hit rates improve by about 9 and 8 percentage points, respectively. DCCC, on the other hand, improves by about 16 percentage points in the same range. The performance gap grows wider with scale. For example, at 8 nodes, DCCC achieves around 81%. ICE is at 71% and DRL-Edge is slightly behind at 68%. The consistent lead of DCCC highlights its advantage in cooperative environments. It uses consistent hashing and deep learning to assign content smartly and distribute requests evenly. This results in higher efficiency and fewer content misses. The figure clearly shows that DCCC scales better than others. It improves cache performance in edge computing. DCCC uses smart cooperation between nodes. This makes it more effective. Distributed reinforcement learning helps it learn better. It performs better than traditional caching methods.

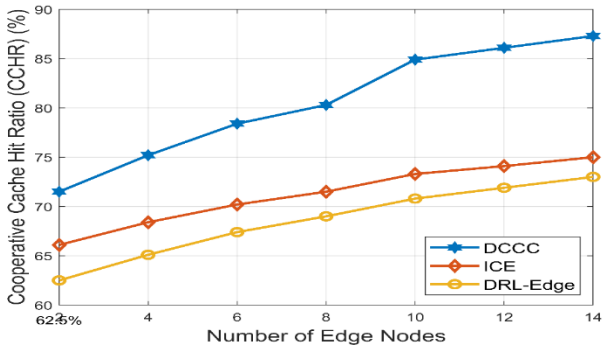

Fig. 5: CCHR Comparison.

Figure 6 shows how cumulative reward changes over 100 training episodes for three methods: DCCC, ICE, and DRL-Edge. In the early episodes, all models improve, but DCCC rises the fastest. It jumps from about 68 to 95 in the first 15 episodes. After that, its growth slows but remains steady. It crosses 100 by episode 20 and reaches around 112 by episode 100. ICE begins at around 60 and climbs at a slower pace. It moves steadily with small dips and ends close to 101. DRL-Edge shows the slowest progress. It starts near 55, grows gradually, and ends below 85. DCCC maintains a clear lead in both learning speed and reward throughout all training episodes. If we look at specific points, DCCC's cumulative reward is already near 100 by episode 25. ICE is around 87, and DRL-Edge is under 80. By episode 50, DCCC stays above 105. ICE fluctuates between 90 and 95. DRL-Edge hovers below 85. In the last 20 episodes, DCCC shows stable and high rewards with fewer drops. ICE shows some rise but with more noise, while DRL-Edge levels off. The gap between DCCC and the others grows over time. This confirms that DCCC learns more efficiently. Its higher and smoother reward curve proves it delivers better caching decisions through distributed coordination and deeper learning.

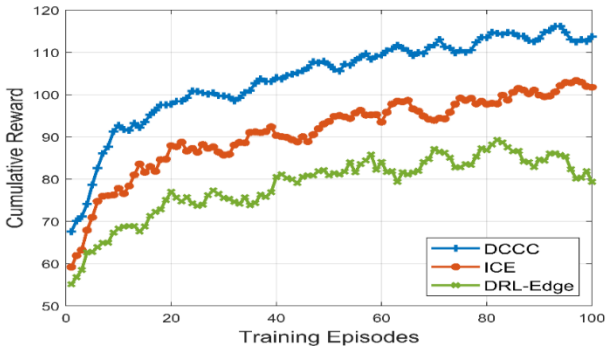

Fig. 6: Learning Curve: Reward Over Episodes.

Figure 7 shows the CHR in percentage for three caching methods—DCCC, ICE, and DRL-Edge. It is under different cache sizes ranging from 100 MB to 700 MB. It starts at 70% for 100 MB and steadily climbs to 88% at 700 MB. ICE follows next, beginning at about 67% and reaching around 81%. DRL-Edge has the lowest CHR, starting at 62% and ending at 74%. As the cache size increases, all methods

show improvement, but the gap between DCCC and the others also widens. Quantitatively, from 100 MB to 400 MB, DCCC improves by 11 percentage points. ICE gains about 9 percentage points. DRL-Edge improves by roughly 9% as well. At 500 MB, DCCC reaches about 85%. ICE is around 79% and DRL-Edge stands at 72%. At the maximum cache size of 700 MB, DCCC achieves the highest CHR of nearly 88% with ICE at 81% and DRL-Edge at 74%. This clearly demonstrates that DCCC gains the most advantage from larger cache sizes. It makes better use of available memory through coordination and intelligent caching strategies. The graph shows that all methods improve with more cache. But DCCC gives the best results. It reaches the highest performance. DCCC scales better in edge caching systems.

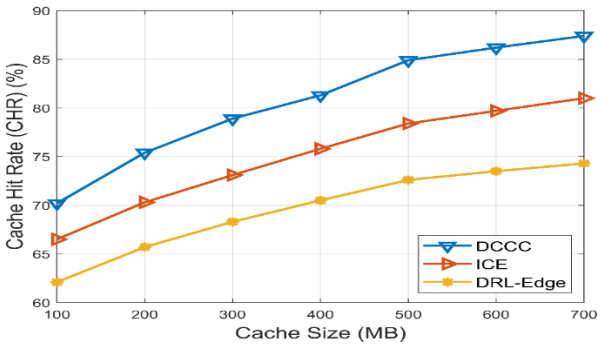

Fig. 7: Effect of Cache Size on CHR.

Figure 8 presents how the average transmission time changes with increasing request rates for DCCC, ICE, and DRL-Edge. As the request rate goes from 10 to 70 requests per second, the delay increases for all methods. DCCC has the lowest delay at the start. It begins at around 110 milliseconds. At 70 requests per second, it reaches about 140 milliseconds. The increase is steady and smooth. ICE begins slightly higher at 120 milliseconds and reaches approximately 158 milliseconds by the end. DRL-Edge shows the highest delay throughout, starting at 132 milliseconds and ending at nearly 177 milliseconds. This shows that DCCC handles network load more efficiently than the other two. The differences between the methods become more noticeable as the request rate increases. At 30 requests per second, DCCC has an average delay of 120 milliseconds. ICE is around 130, and DRL-Edge is above 140. At 60 requests per second, DCCC maintains a delay close to 135 milliseconds. ICE reaches about 150, and DRL-Edge approaches 170. The gap widens under heavy traffic, proving that DCCC is more stable and scalable. Its slower increase in delay suggests smarter decision-making and better caching efficiency. This makes DCCC a strong choice for systems that need to serve content quickly, even when many requests are made at once.

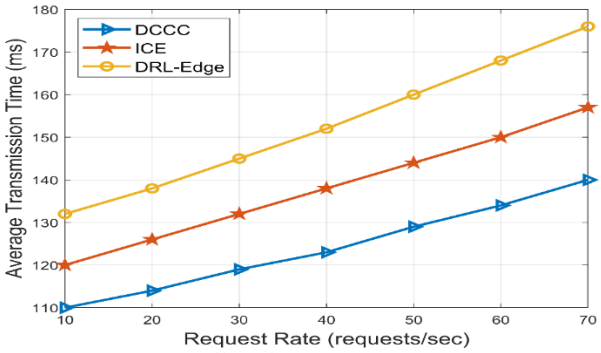

Fig. 8: Effect of Request Rate on ATT.

Figure 9 shows the content exchange time for four caching strategies: ICE, AC, LRU, and LFU. Each method is evaluated based on its minimum, average, and maximum content exchange time. ICE has the lowest values among all methods. Its minimum exchange time is around 5000. The average is about 6000. The maximum is just above 6200. AC comes next with higher values. Its minimum is 7000. The average is around 8500. The maximum reaches 9000. In comparison, LRU and LFU show similar and slightly higher exchange times than AC. LRU's values are approximately 7500 (min), 8000 (avg), and 8700 (max). LFU shows similar results to AC. It starts at about 7300. The average is around 8100. The maximum goes up to 8800. ICE has the lowest overhead. It handles content coordination better. This makes ICE more efficient for node communication. AC, while adaptive, introduces more frequent content shifts. LRU and LFU show the highest load in content exchange. It is due to static policies causing frequent replacements. The chart clearly indicates that smarter and cooperative strategies like ICE reduce content exchange overhead significantly.

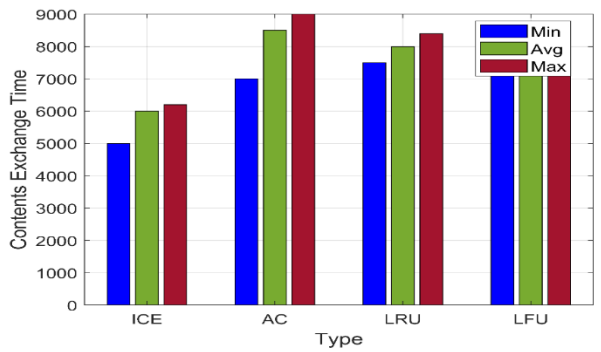

Fig. 9: Content Exchange Time for Different Caching Types.

Figure 10 shows how average energy consumption (in joules) changes with increasing cache size for three caching methods: DCCC, ICE, and DRL-Edge. All three methods consistently show a decrease as the cache size increases. DCCC begins at around 2.8 J with 100 MB and decreases steadily to just below 1.9 J at 700 MB. ICE starts higher, at 3.0 J, and reduces to about 2.0 J. DRL-Edge consumes the most energy, starting at 3.2 J and dropping to approximately 2.2 J. Quantitatively, DCCC shows the best energy efficiency throughout. Between 100 MB and 400 MB, DCCC's energy usage falls by about 0.6 J. ICE reduces by about 0.7 J, and DRL-Edge by nearly 0.6 J. However, at every point, DCCC stays ahead with the lowest energy values. At 500 MB, DCCC is around 1.95 J. ICE is at 2.15 J, and DRL-Edge is at 2.3 J. By 700 MB, DCCC reaches its lowest point just below 1.9 J. ICE ends near 2.0 J, and DRL-Edge is at 2.2 J. These results prove that DCCC is more energy-efficient and scales better as cache size increases. The smoother drop in DCCC's curve shows it uses cache space more intelligently. It reduces data transmission and energy usage. This underscores DCCC's strength in optimizing resource usage in caching systems.

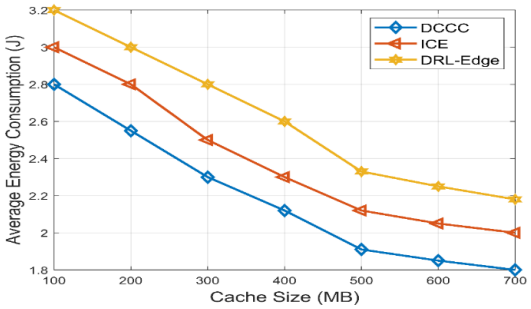

Fig. 10: Energy Consumption Versus Cache Size.

Figure 11 shows the loss curves with clear differences in learning behavior across 100 training episodes. DCCC performs better with the lowest loss values throughout the training. ICE comes next, showing steady improvement but slightly higher values than DCCC. DRL-Edge has the highest loss and the most fluctuations. It indicates slower and less stable learning. Early in the training, all models start with high loss. The gap between DCCC and the others widens quickly. DCCC's curve is smoother, showing better convergence. DRL-Edge has more noise and variability. By looking at the numbers, we see that in the first 20 episodes, DCCC reduces its loss from around 1.0 to below 0.6. ICE follows with a drop from 1.15 to around 0.7. DRL-Edge lags still near 0.85. At episode 50, DCCC is close to 0.3. ICE near 0.4 and DRL-Edge slightly above 0.5. Toward the end of training, DCCC reaches its lowest point near 0.1. ICE stabilizes around 0.2, and DRL-Edge remains above 0.3. These observations indicate that DCCC learns more quickly, adapts more effectively, and delivers more reliable training results. Its lower and smoother loss curve demonstrates stronger model stability and more efficient optimization.

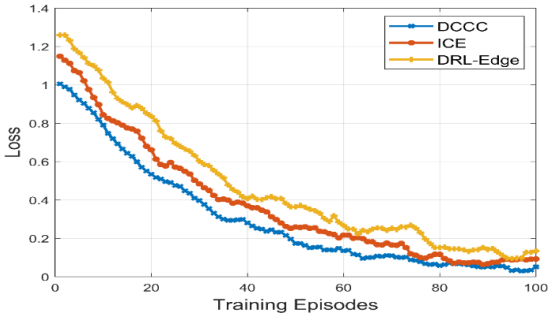

Fig. 11: Loss Over Episodes for Different Caching Methods.

Figure 12 shows the Q-value variance over 100 training episodes for three caching models: DCCC, ICE, and DRL-Edge. A lower variance means better learning stability. At the start, all models begin with high variance. DRL-Edge starts at the top with around 0.39. ICE begins at about 0.28 and DCCC at around 0.24. As training continues, the variance decreases for all methods, but the rate and smoothness differ. DCCC shows the fastest and smoothest decline. ICE reduces steadily as well. DRL-Edge shows more fluctuation throughout the training process. By episode 50, DCCC's Q-value variance has dropped below 0.10. ICE is around 0.12, and DRL-Edge is still around 0.17. As the training moves forward, DCCC continues to improve and maintains a variance near 0.03 toward the end. ICE stabilizes and ends close to 0.04. DRL-Edge shows more delay. It ends with a higher variance, near 0.06. This suggests that DCCC learns more efficiently and converges to a stable policy faster than the other methods. The smoother and lower curve of DCCC proves that it offers better training stability. ICE performs well. DRL-Edge struggles with learning. It has a higher variance in results. This happens more in the early and middle episodes. Its learning is less stable. This figure shows that DCCC not only learns faster but is also more confident and reliable across the training episodes.

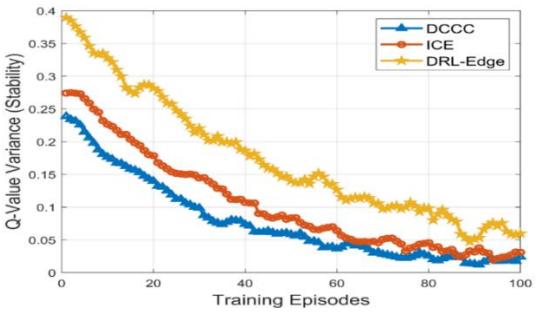

Fig. 12: Q-Value Stability Over Episodes.

In Figure 13, as the number of edge nodes increases, the CCHR increases for all four caching methods—DCCC, AC, LRU, and LFU. Among these, DCCC demonstrates the most significant increase. It begins at a higher level than the others and continues to grow more rapidly with each additional node. AC improves steadily but stays below DCCC throughout. LRU and LFU begin with lower values and only gain slightly as the number of nodes increases. DCCC maintains the highest performance. The traditional methods, like LRU and LFU, show limited benefits from added nodes. This reflects how advanced coordination strategies enhance caching performance in large systems. Looking at the numbers, DCCC grows from 70% at 2 nodes to 88% at 14 nodes. It gains 18 percentage points. AC increases from 65% to 80% a gain of 15 points. LFU starts at 56% and ends at 67%. LRU begins at 55% and reaches 65%. At 10 nodes, DCCC is already around 85%. AC is at 77%, LFU near 64% and LRU just above 63%. These values show that DCCC uses cooperation more effectively. It scales well as more nodes are added, giving higher hit ratios. In contrast, LRU and LFU do not take much advantage of distributed caching. It makes them less suitable for cooperative edge environments. The figure confirms the benefit of smart caching. Intelligent coordination improves performance. Learning methods make caching more adaptive. This makes it more efficient. These methods scale well. They work better as more edge nodes are added.

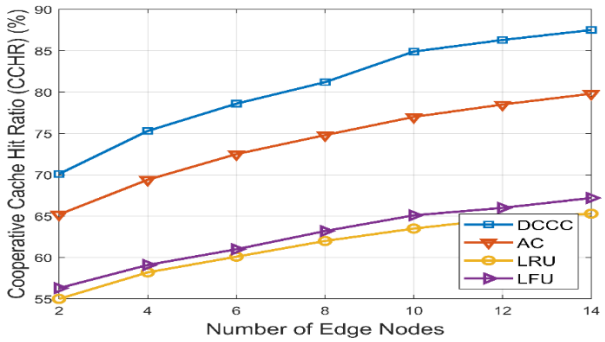

Fig. 13: CCHR Versus Number of Edge Nodes.

Figure 14 illustrates how the CHR changes as cache size increases from 100 MB to 700 MB for four caching methods. As the cache size increases, all four methods show improvements in hit rate, but at different rates. DCCC performs the best starting at about 71% with 100 MB. It is reaching nearly 88% at 700 MB. AC follows, starting at around 65% and ending at 80%. LFU and LRU start lower, both near 55%, 57% rise more slowly. It ends around 67% and 65%, respectively. Quantitatively, DCCC improves by 17 percentage points over the full cache size range. AC improves by 15 points. LFU and LRU show smaller gains of about 12 and 10 points. At 300 MB, DCCC already reaches around 78%. AC is near 72%, LFU is around 61% and LRU is close to 60%. At 500 MB, DCCC climbs above 84%. AC reaches 77%, LFU is around 63% and LRU is near 62%. This shows that DCCC scales better with more cache space and makes better caching decisions. AC performs reasonably well but falls behind DCCC. Traditional methods like LRU and LFU show slower growth and lower performance overall. The figure clearly shows the benefit of DCCC. It uses larger cache sizes more effectively. DCCC gives higher hit rates. It performs better than basic caching methods. Learning-based strategies have a clear advantage.

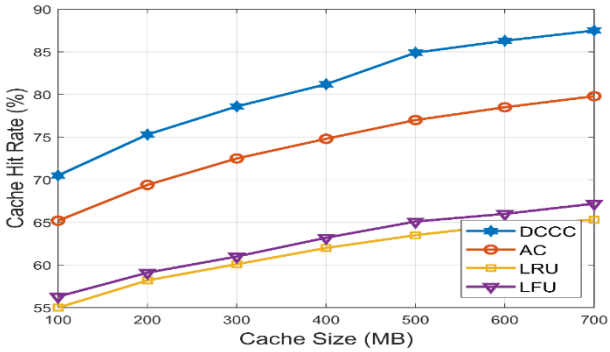

Fig. 14: Cache Hit Rate Comparison.

Figure 15 shows how the energy saved (%) increases with the number of edge nodes for four caching methods. As the number of nodes increases, all four methods show improvement, but at different rates. DCCC performs best. It starts at 20% energy saved with 2 nodes and climbs steadily to about 78% with 14 nodes. AC shows strong gains, beginning at 15% and reaching around 68% by the end. LFU and LRU show much slower growth. LFU starts at 10% and ends close to 30%. LRU begins at 8% and finishes just above 22%. Quantitatively, DCCC shows an increase of 58 percentage points from 2 to 14 nodes. It is the highest among all methods. AC increases by 53 percentage points in the same range. LFU gains about 20 percentage points. LRU shows the smallest improvement of roughly 14 points. At 8 edge nodes, DCCC already saves about 60% energy. AC is close to 50%, LFU around 21% and LRU about 18%. The performance gap widens as more nodes are added between DCCC and the traditional methods. This result clearly shows that DCCC uses cooperation more effectively to reduce energy use. It adapts better to larger networks, resulting in higher energy savings. In contrast, LRU and LFU do not benefit much from added nodes. It suggests that basic caching techniques are less efficient in dynamic, distributed edge environments.

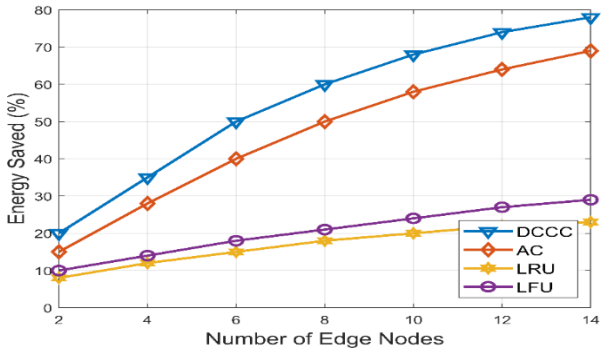

Fig. 15: Energy Saved Versus Number of Nodes.

In Figure 16, by the end of 100 training episodes, DCCC achieves the highest cache hit rate. It reaches around 85%. ICE follows with a final value of about 79%. DRL-Edge ends at approximately 74%. These results show that DCCC offers better caching accuracy than the other two methods. The performance gap between DCCC and DRL-Edge is more than 10 percentage points. The difference between DCCC and ICE is around 6 points. This shows that DCCC delivers stronger and more effective learning in dynamic environments. DCCC starts at 62%. It rises sharply in the first 30 episodes. By episode 40, it goes above 80%. The increase is fast and smooth. ICE begins at 60% and grows steadily, reaching 75% near episode 50 before slightly slowing down. DRL-Edge starts lower at 57% and takes longer to improve, reaching just above 70% after 60 episodes. Its curve is less smooth with more fluctuations throughout. DCCC, on the other hand, maintains a steady increase with fewer fluctuations, indicating stable learning. The overall pattern suggests that DCCC is more efficient and consistent. ICE performs moderately, and DRL-Edge improves more slowly and remains behind in hit rate.

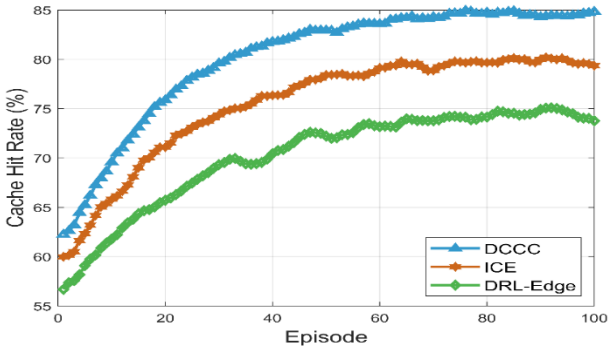

Fig. 16: Episode-Wise Cache Hit Rate Comparison.

7. Conclusion

This paper proposed two intelligent caching systems for edge networks: ICE and DCCC. The main goal was to reduce transmission delay and save energy by storing useful content closer to users. Both systems used DRL to make smart caching decisions. The first system, ICE, works on a single edge node. It uses a model based on Newton's Law of Cooling to track how content popularity changes over time. ICE learns which files are valuable by observing user requests. It then decides which content to keep or remove from the cache. Over time, ICE improves and becomes better at storing the most needed content. The second system, DCCC, is for multiple edge nodes working together. It uses consistent hashing to divide content among the nodes. This avoids storing the same content in many places. DCCC balances the load by sending user requests to nearby nodes when one node is busy. This cooperation improves the use of storage and reduces overall system delay. The work showed that both ICE and DCCC are flexible. They work well even when content demand changes. They adjust to different cache sizes and user request rates. This makes them useful for many real-world edge computing environments, for example, 5G, smart cities, and IoT systems.

References

[1] K. Shafique, B. A. Khawaja, F. Sabir, S. Qazi M. Mustaqim, "Internet of things (iot) for next-generation smart systems: A review of current challenges, future trends and prospects for emerging 5g-iot scenarios," IEEE access, vol. 8, pp. 23022–23040, 2020. https://doi.org/10.1109/ACCESS.2020.2970118.

[2] Y. Zhang and A. ˚Arvidsson, "Understanding the characteristics of cellular data traffic," in Proceedings of the 2012 ACM SIGCOMM workshop on Cellular networks: operations, challenges future design, pp. 13–18, 2012. https://doi.org/10.1145/2342468.2342472.

[3] S. Crites, "The narrative quality of experience," Journal of the American academy of religion, vol. 39, no. 3, pp. 291–311, 1971. https://doi.org/10.1093/jaarel/XXXIX.3.291.

[4] N. P. Lago and F. Kon, "The quest for low latency," in ICMC, 2004.

[5] W. Yu, F. Liang, X. He, W. G. Hatcher, C. Lu, J. Lin X. Yang, "A survey on the edge computing for the internet of things," IEEE access, vol. 6, pp. 6900–6919, 2017. https://doi.org/10.1109/ACCESS.2017.2778504.

[6] A. Liu, M. Huang, M. Zhao T. Wang, "A smart high-speed backbone path construction approach for energy and delay optimization in wsns," IEEE Access, vol. 6, pp. 13836–13854, 2018. https://doi.org/10.1109/ACCESS.2018.2809556.

[7] T. Wang, Y. Deng, J. Mao, M. Chen, G. Liu, J. Di K. Li, "Towards intelligent adaptive edge caching using deep reinforcement learning," IEEE Transactions on Mobile Computing, vol. 23, no. 10, pp. 9289–9303, 2024. https://doi.org/10.1109/TMC.2024.3361083.

[8] A. Mondol, R. Gupta, S. Das T. Dutta, "An insight into newton's cooling law using fractional calculus," Journal of Applied Physics, vol. 123, no. 6, 2018. https://doi.org/10.1063/1.4998236.

[9] K.-W. Zhao, W.-H. Kao, K.-H. Wu Y.-J. Kao, "Generation of ice states through deep reinforcement learning," Physical Review E, vol. 99, no. 6, p. 062106, 2019. https://doi.org/10.1103/PhysRevE.99.062106.

[10] W. K. Chai, D. He, I. Psaras G. Pavlou, “Cache “less for more” in information-centric networks (extended version),” Computer Communications, vol. 36, no. 7, pp. 758–770, 2013. https://doi.org/10.1016/j.comcom.2013.01.007.
[11] X. Zhou, F. Shen, L. Liu, W. Liu, L. Nie, Y. Yang H. T. Shen, “Graph convolutional network hashing,” IEEE transactions on cybernetics, vol. 50, no. 4, pp. 1460–1472, 2018. https://doi.org/10.1109/TCYB.2018.2883970.
[12] R. Dadi, K. Meenakshy S. Damodaran, “A review on secondary control methods in dc microgrid,” Journal of Operation and Automation in Power Engineering, vol. 11, no. 2, pp. 105–112, 2023.
[13] X. Xia, F. Chen, Q. He, J. Grundy, M. Abdelrazek H. Jin, “Online collaborative data caching in edge computing,” IEEE Transactions on Parallel and Distributed Systems, vol. 32, no. 2, pp. 281–294, 2020. https://doi.org/10.1109/TPDS.2020.3016344.
[14] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher V. Young, “Mobile edge computing—a key technology towards 5g,” ETSI white paper, vol. 11, no. 11, pp. 1–16, 2015.
[15] W.-X. Liu et al., “Content popularity prediction and caching for ICN: A deep learning approach with SDN,” IEEE Access, vol. 6, pp. 5075–5089, 2017. https://doi.org/10.1109/ACCESS.2017.2781716.
[16] Y. Im et al., “SNN-cache: A practical machine learning-based caching system utilizing the inter-relationships of requests,” in Proc. CISS, 2018, pp. 1–6. https://doi.org/10.1109/CISS.2018.8362281.
[17] K. C. Tsai et al., “Mobile social media networks caching with convolutional neural network,” in Proc. IEEE WCNC Workshops, 2018, pp. 83–88. https://doi.org/10.1109/WCNCW.2018.8368988.
[18] A. Lekharu et al., “Deep learning model for content aware caching at MEC servers,” IEEE Trans. Netw. Service Manag., vol. 19, no. 2, pp. 1413–1425, 2022. https://doi.org/10.1109/TNSM.2021.3136439.
[19] Y. Zhang et al., “PSAC: Proactive sequence-aware content caching via deep learning at the network edge,” IEEE Trans. Netw. Sci. Eng., vol. 7, no. 4, pp. 2145–2154, 2020. https://doi.org/10.1109/TNSE.2020.2990963.
[20] N. Kumar et al., “A novel distributed Q-learning based resource reservation framework for facilitating D2D content access requests in LTE-a networks,” IEEE Trans. Netw. Service Manag., vol. 15, no. 2, pp. 718–731, 2018. https://doi.org/10.1109/TNSM.2018.2807594.
[21] S. Liu et al., “Distributed reinforcement learning for privacy-preserving dynamic edge caching,” IEEE J. Sel. Areas Commun., vol. 40, no. 3, pp. 749–760, 2022. https://doi.org/10.1109/JSAC.2022.3142348.
[22] H. Zheng et al., “Reinforcement learning for energy-efficient edge caching in mobile edge networks,” in Proc. IEEE INFOCOM Workshops, 2021, pp. 1–6. https://doi.org/10.1109/INFOCOMWKSHPS51825.2021.9484635.
[23] R. Wang et al., “Cooperative caching strategy with content request prediction in Internet of Vehicles,” IEEE Internet Things J., vol. 8, no. 11, pp. 8964–8975, 2021. https://doi.org/10.1109/JIOT.2021.3056084.
[24] C. Zhong et al., “Deep reinforcement learning-based edge caching in wireless networks,” IEEE Trans. Cogn. Commun. Netw., vol. 6, no. 1, pp. 48–61, 2020. https://doi.org/10.1109/TCCN.2020.2968326.
[25] L. T. Tan and R. Q. Hu, “Mobility-aware edge caching and computing in vehicle networks: A deep reinforcement learning,” IEEE Trans. Veh. Technol., vol. 69, no. 10, pp. 11303–11317, 2020.
[26] X. Chen et al., “Distributed edge caching for dynamic contents in wireless service,” IEEE Trans. Wireless Commun., vol. 18, no. 1, pp. 518–531, 2019.
[27] L. Qian et al., “Collaborative edge caching with content placement in user-centric networks,” IEEE Access, vol. 8, pp. 141512–141522, 2020.
[28] J. Shi et al., “Proactive cooperative caching in mobile edge networks: A deep learning approach,” in Proc. IEEE GLOBECOM, 2020.
[29] B. He et al., “Multi-agent AC-based collaborative caching in vehicular networks,” IEEE Internet Things J., vol. 7, no. 6, pp. 4817–4827, 2020.
[30] Q. Wang et al., “Online collaborative edge caching with Lyapunov optimization,” IEEE Trans. Veh. Technol., vol. 69, no. 6, pp. 6822–6835, 2020.
[31] Satyam A., Kumar R.A., Patchala S., Pachala S., Atkar G.B., Mahalaxmi U.S.B.K. “Multi-agent learning for UAV networks: a unified approach to trajectory control, frequency allocation and routing” International Journal of Basic and Applied Sciences, 14 (2), pp. 189 - 201, 2025. https://doi.org/10.14419/474dfq89.
[32] Kailasam N., Yalamati S., Murthy V.S.N., Venkateswara Rao P., Anil Kumar R., Jayaram Kumar K. “Optimized Task Offloading in D2D-Assisted Cloud-Edge Networks Using Hybrid Deep Reinforcement Learning”International Journal of Basic and Applied Sciences, 14 (2), pp. 591 - 602, 2025. https://doi.org/10.14419/xm2ebp25.
[33] Uusitalo, Mikko A., Mårten Ericson, Björn Richerzhagen, Elif Ustundag Soykan, Patrik Rugeland, Gerhard Fettweis, Dario Sabella et al. "Hexa-X the European 6G flagship project." In 2021 Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit), pp. 580-585. IEEE, 2021. https://doi.org/10.1109/EuCNC/6GSummit51104.2021.9482430.
[34] Yakushiji, Koki, Hiroshi Fujita, Mikio Murata, Naoki Hiroi, Yuuichi Hamabe, and Fumiatsu Yakushiji. "Short-range transportation using unmanned aerial vehicles (UAVs) during disasters in Japan." Drones 4, no. 4, 2020. https://doi.org/10.3390/drones4040068.
[35] Ummiti Sreenivasulu, Shaik Fairooz, R. Anil Kumar, Sarala Patchala, R. Prakash Kumar, Adireddy Rmaesh, “Joint beamforming with RIS assisted MU-MISO systems using HR-mobilenet and ASO algorithm, Digital Signal Processing, Volume 159, 2025, 104955,ISSN 1051-2004, https://doi.org/10.1016/j.dsp.2024.104955.
[36] Satyam, A. ., Kumar, R. A. ., Patchala , S. ., Pachala, S. ., Geeta Bhimrao Atkar, & Mahalaxm, U. S. B. K. . (2025). Multi-agent learning for UAV networks: a unified approach to trajectory control, frequency allocation and routing. International Journal of Basic and Applied Sciences, 14(2), 189-201. https://doi.org/10.14419/474dfq89.
[37] Kumar , B. P. ., Mahalaxmi , U. S. B. K. ., Nagagopiraju, V. . ., Manda , A. K. ., Chandana , K. ., Betam, D. Suresh . ., Gangadhar , A. ., & Patchala, D. S. . . (2025). Deep Reinforcement Learning for Joint UAV Trajectory and Communication Design in Cache-Enabled Cellular Networks. International Journal of Basic and Applied Sciences, 14(3), 418-430. https://doi.org/10.14419/djn77m90.