Reinforcement Learning-Based Resource Management for Crowd-sourced Streaming Video Services

Haijian Wu, Mai Pham, Wenhan Yang, Yijing Xu
Columbia University in the City of New York

Abstract—This project addresses the challenge of optimizing resource allocation on live-streaming video platforms. It proposes a sophisticated approach that combines Machine Learning-Deep Q-Learning (ML-DQL), Double Deep Q-Network (Double DQN), and Count-based Exploration DQN. The focus is on solving the problem of efficiently distributing server resources amidst the dynamically changing demand of users, ensuring both high-quality streaming experience and cost-effectiveness. The paper's findings demonstrate significant improvements in resource utilization, offering valuable insights for managing complex, user-driven live-streaming environments.

Index Terms—Geo-distributed Clouds, Reinforcement Learning, Machine Learning, Live-Streaming Platforms, Resource Allocation.

1 Introduction

Live streaming services have revolutionized digital media, driven by advancements in technology, content platforms like Facebook and YouTube, and widespread internet access. This transformation has led to a novel era of media interaction, though not without challenges. Crowdsourced live streaming must reconcile user experience, or Quality of Experience (QoE), with operational costs. The service's success hinges on its adaptability to the unpredictable nature of content creation and consumption, and managing latency is crucial for preserving real-time interaction and high QoE.

Participants in live streaming, both content creators and consumers, exhibit a broad spectrum of preferences and technological capabilities. Addressing this diversity and the volatile nature of user engagement demands a resilient and adaptive framework for content delivery. Balancing these diverse requirements with the unpredictability of live streaming is key to improving service responsiveness and effectiveness.

Our project presents a reinforcement learning-based solution tailored to address the dynamic demands in live streaming. The approach emphasizes latency reduction, content personalization, and cost-QoE optimization, showcasing the potential of our model to enhance streaming efficiency and adapt to the diverse needs of a global audience, thereby advancing live streaming technologies.

1.1 Background and Related Work

In the realm of content providers (CPs) like YouTube and Netflix, and with the advent of advanced gaming platforms and multimedia technologies, live streaming has surged in popularity, drawing significant attention from both academia and industry. Most previous work can be categorized into two systems – optimization and prediction.

FIn the study by Bilal et al. [1], a greedy algorithm known as GMC was developed for the selection of cloud sites tailored for video transcoding. This algorithm's goal is to reduce operational expenses and enhance the Quality of Experience (QoE). Nevertheless, it does not account for the broad geographical distribution of viewers. On the other hand, the research in [2] introduces a comprehensive optimization strategy for choosing data centers and distributing video streams, with the dual objectives of lowering network costs and streamlining content delivery delays. This study, however, assumes a static distribution of viewers and a consistent popularity of videos throughout the broadcast, a scenario not always applicable to crowdsourced content dynamics. Similarly, the authors in [3] focus on an optimization model that aims to determine the ideal quantity of cloud data centers to facilitate cost-effective video migration by content crowdsourcers.

Meanwhile, Haouari et al. [4] and Yuan et al. [5] have put forth predictive models that estimate future resource demands. Subsequently, real-time video allocation is executed based on heuristic methods within the bounds of these forecasted resources. However, these models emphasize the need to analyze content popularity to better utilize the resources at hand. Decisions in these models are often made in response to the immediate state of cloud sites. Furthermore, most existing literature presumes static cloud platform configurations, such as a fixed number of sites and predefined cost and delay parameters. This stands in contrast to our Reinforcement Learning (RL) methodology, which seeks optimal solutions and is capable of adapting to changes in the environment through ongoing online learning. From [6], the author proposed reinforcement learning architectures with DQN, but they lack further exploration and highly focus on machine learning predictions.

1.2 Motivation

From related work, developing an effective resource allocation strategy becomes not difficult when user demand and

the count of live video streams remain constant. However, the real complexity arises in crafting an online algorithm capable of leveraging cloud resources to accommodate the unpredictable influx of live video content and manage the dynamic, geographically dispersed user base instantaneously. Additionally, this algorithm strives to approximate the optimal solutions that are typically achieved through offline optimization, benefiting from a comprehensive, long-term perspective of the system's behavior.

1.2.1 Trade-off between Cost and Quality of Experience (QoE)

Live streaming services are uniquely sensitive to the interaction delay between viewers and broadcasters, such as the latency in online comments and reactions. This delay is a critical factor that significantly influences the Quality of Experience (QoE). Therefore, it is imperative to maintain the average delay below a certain threshold required by the application to ensure optimal viewer satisfaction. Exploring higher thresholds provides insights into the cost-QoE trade-off. This project, based on literature review, focuses exclusively on the round-trip delay across various data centers and aims to minimize delivery delays to enhance QoE when keeping the cost not too high. The strategic contribution of this work lies in selecting the best data centers for content migration and replication for transcoding purposes, as well as identifying proximate sites to serve viewers at a minimized cost. The period T is divided into equal intervals t, with $t = 0$ marking the starting time slot. Let $V(t) = \{v_1, v_2, v_3, \ldots, v_m\}$ denote the set of live videos broadcast during interval t. It is assumed that all content shares a uniform size Z_v 0.738 GB, and the time interval t suffices to upload the entire stream to the system. For the delay between every two data centers, we use a matrix that includes constant delay with the unit of ms. The trade-off here is the major matrix for us to consider our reward function in section 2.

1.3 Machine Learning Prediction for Viewers

In this project, Facebook's live streaming service is utilized, analyzing a Facebook dataset [7] with millions of video streams. Our preprocessing involves selecting immediate features like broadcast time and location, associating viewers with the closest AWS cloud site, and utilizing feature hashing and one-hot encoding for categorical data. We reduce the high-dimension features like broadcaster id and categories into ten features for each. For the broadcasting time features, we also map them based on day (from Sunday to Saturday) and time intervals (every four hour, starting from 0:00).

We predict the number of viewers using regression models based on Random Forest, XGBoost, and MLP algorithms. The efficacy of these models is quantified by the R^2 metric, as follows:

$$R^2 = 1 - \frac{\sum_{i=1}^{V}(r_i - p_i)^2}{\sum_{i=1}^{V}(r_i - \bar{r})^2} \tag{1}$$

where V represents the total number of video records, r_i and p_i are the actual and predicted viewers, and $\bar{r}$ is the average viewership. A higher R^2 value indicates greater predictive accuracy. Based on our models, random forests outperform a lot than XGBoost and MLP. Random forests achieved 91.45%. This is also consistent with the findings from [6].

2 Formulation of Reinforcement Learning Problem

2.1 State

At each time-step t_s, the state is a tuple $S(t_s) = (r_s^k, p(v_i, r_s^k), \text{Cost}, \text{IAD})$, where r_s^k is the data center index, $p(v_i, r_s^k)$ denotes the predicted number of viewers at data center r_s^k for video v_i, Cost is the accumulated cost from previous steps, and IAD (Incremental Average Delay) represents the delay accumulated over time.

- r_s^k data center index: We design our system to be composed of 10 AWS sites, namely US West-California, US East-Virginia, US East-Ohio, South America-Sao Paulo, Europe-Paris, Europe-Frankfurt, China-Ningxia, Asia-Singapore, Asia-Seoul, and Asia-Mumbai.
- $p(v_i, r_s^k)$ predicted number of viewers at data center r_s^k for video v_i: this refers to the estimated number of viewers expected to watch a particular video stream at a given time and data center locations. In practice, the number of viewers for future timesteps may not be known, so it's important to simulate the data with machine learning for modeling. The simulation and machine learning models have been discussed in Section 1.

The system updates Cost and IAD at each timestep t_s based on predicted action using ML. The updated cost and delay become the input for the next timestep t_{s+1}. The goal is to maximize total reward as typical reinforcement learning.

2.1.1 Cost

In the context of resource allocation for live video streaming, our model categorizes costs into three distinct types, each critical for optimizing resource usage and maintaining service quality:

1) **Rental/Server Cost** c_j^r : This cost is incurred from the utilization of server resources for each video v_i, associated with the data center (or cloud platform). We assume that the provisioning of resources is based on on-demand charging. This cost varies depending on the cloud site and the data usage level fixed by the cloud platform. It is calculated based on factors like server utilization and operational expenses for processing and delivering video content. We can find related information from https://aws.amazon.com/.

$$Cost_r(t) = \sum_{v_i \in V(t)} \sum_{r_j \in R} c_j^r(t) \cdot Z_{v_i} \cdot H(v_i, r_j)$$

where $c_j^r(t)$ indicates the cost associated with the j-th data center at time t, Z_v is a the size of video v_i, and $H(v_i, r_{a_j})$ is a function that determines whether

the video v_i is allocated to the j-th data center in region r_j.

2) **Migration/Switching Cost** ($c_{r_a}^{r_b}$): Arising when the streaming data center is changed, this cost reflects the expenses involved in rerouting data and adjusting resource allocation during the transition from r_a to r_b. It is a crucial factor in managing the dynamics of data center utilization. When a viewer requests a video from his/her closest data center, which is not currently serving the content, the request is either satisfied from a far-away cloud or migrated to a closer one in order to meet the overall latency requirement of the system.

$$Cost_M(t) = \sum_{v_i \in V(t)} \sum_{r_a \in R_g} \sum_{r_b \neq r_a \in R_g} c_{r_b}^{r_a}(t) \cdot Z_v \cdot H(v_i, r_j)]$$

where $c_{r_a}^{r_b}$is the price of migrating a copy of the content v_i from the data center from r_a to the target data center r_b per GB. There is a price matrix, and this cost is zero if no migration is needed.

3) **Serving/Service Cost** (c_j^s): When distributed viewers download contents from selected cloud sites, the CP should pay the cost of serving these requests. The bandwidth cost is presented as follows:

$$Cost_S(t) = \sum_{v_i \in V(t)} \sum_{r_j \in R_g} c_j^s \cdot Z_{v_i} \cdot p(v_i, r_j) \cdot H(v_i, r_j)$$

where $p(v_i, r_j)$ is the number of predicted viewers of video v_i at data center r_j.

For each video v_i at timestep t_s, the total cost is the summation of them

$$Cost(t) = Cost_R(t) + Cost_M(t) + Cost_S(t)$$

For the initial timestep, we assume the default cost is only the migration cost from the broadcaster location to a random location.

2.1.2 IAD Incremental Average Delay

IAD denote the Incremental Average Delay of different timesteps in one episode. Particularly, IAD defines the average delay of the current time-step ts and previous timesteps, meaning the sum of $delay_1, ..., delay_{t_s}$. At each t_s, we test if the incremental average delay respects the threshold. When t_s = N^{v_i} for each video v_i, IAD is equal to the total average delay to serve all viewers in all regions.

2.2 Action

Actions in this model might involve selecting specific data centers, allocating bandwidth, or other decisions related to resource distribution to optimize streaming quality.

In the proposed framework, the objective of the reinforcement learning (RL) agent is to minimize operational costs while simultaneously maximizing the Quality of Experience (QoE) for viewers. At each time-step t_s, the agent determines the most suitable cloud site to serve the predicted number of viewers $p(v_i, r_k^s)$ in region r_k^s. This decision-making process is represented by an action space $A(t_s)$, expressed as a vector (e.g., $[0, 1, \ldots, 0, 0]$), where each element corresponds to a potential cloud site, and in our case there are 10 data centers. A '1' in this vector signifies the selected site for service. If $p(v_i, r_k^s) = 0$ (indicating no predicted viewers). When an element r_j^a in the action vector is '1', it implies that $H(v_i, r_j^a)$ is also set to '1', thereby activating the site for streaming service. In our implementation, we use a single integer to replace the vector in order to keepe the codes easy to understand.

2.3 Reward

The reward function is designed to maximize viewer satisfaction and streaming quality while minimizing costs. It takes into account factors like streaming quality, cost-efficiency, and system stability.

$$R(s, a, s') = w_{\text{cost}} \cdot \Delta C(s, a, s') + w_{\text{QoE}} \cdot \Delta \text{QoE}(s, a, s')$$

Where:

- $R(s, a, s')$ denotes the reward for an action a in state s leading to state s'.
- $\Delta C(s, a, s')$ quantifies the cost change resulting from action a transitioning from state s to s'.
- $\Delta \text{QoE}(s, a, s')$ measures the QoE improvement due to action a from state s to s'.
- w_{cost} and w_{QoE} are weights to balance cost reduction and QoE improvement.

2.3.1 Cost Reduction

$$\Delta C(s, a, s') = C(s) - C(s')$$

Where:

- $C(s)$ is the initial cost in state s.
- $C(s')$ is the cost in the new state s' post-action a.

The cost calculation formula is from Section 2.1.1.

2.3.2 Improvement of QoE

The reward design for QoE is based on adherence to a set of constraints used in optimizations from our literature research designed to ensure efficient operation of the geo-distributed cloud platform. The constraints are defined as:

$$\begin{cases} \text{C1:} & \text{if } p(v_i, r_s^k) > 0 \text{ then } \sum A(t_s) = 1, \\ \text{C2:} & \text{if } p(v_i, r_s^k) = 0 \text{ then } \sum A(t_s) = 0, \\ \text{C3:} & \text{IAD} + \text{delay}_{t_s} \leq D, \end{cases} \tag{2}$$

We articulate the reward for improvement of QoE as follows:

- 0 is given if either the predicted viewers are no zero and the sum of actions is 0, or if the predicted viewers are zero and any different data center is allocated. This approach ensures resource allocation is strictly in line with demand, preventing wastage.
- 1is granted when all constraints are met, indicating an ideal alignment of resources with viewer requirements and maintaining the Incremental Average Delay (IAD) below the defined threshold.
- If the first two constraints (C1 and C2) are satisfied but the IAD exceeds the threshold (C3 is violated), a deferred reward is accounted for in the variable dr. This deferred reward is added to the total reward if, at the conclusion of the video v_i (when $t_s = N^{v_i}$),

the IAD is below the threshold, thus balancing immediate and long-term performance.

The constraints that govern the reward calculation are encapsulated as follows:

$$\begin{cases} 0 & \text{if C1, C2 not satisfied,} \\ 1 & \text{if C1, C2, C3 satisfied,} \\ dr = dr + 1 & \text{if C1, C2 satisfied, C3 not satisfied.} \end{cases} \quad (3)$$

3 ALGORITHM

All Algorithms pseudo-codes and hyper-parameters are shown in appendix except for greedy algorithm, which is our baseline model and very straightforward as follows.

The Greedy algorithm implemented in our study employs a straightforward yet intuitive approach to video allocation in a cloud-based streaming environment. At the commencement of a streaming session, the algorithm allocates a video to the data center that is anticipated to have the highest number of viewers, based on predictive models. In this context, we choose the data center at California (U.S. West). This decision is made once and remains static for the duration of the video streaming session, with no subsequent reallocation actions regardless of any changes in viewer distribution that may occur over time. The simplicity of this approach lies in its one-time decision-making process at the start, focusing on maximizing the immediate viewer reach by leveraging the predicted concentration of viewers for optimal initial placement. However, due to its non-adaptive nature, this method does not account for dynamic shifts in viewer behavior or distribution after the initial allocation, in contrast to the following reinforcement learning algorithms.

3.1 DQN

Deep Q-Networks (DQN) are a breakthrough approach in the field of artificial intelligence, specifically within a domain known as reinforcement learning. Reinforcement learning involves training an 'agent' to make decisions: the agent learns to perform actions in an 'environment' to achieve a goal, guided by rewards for correct actions. Traditional methods face challenges when dealing with complex environments where the potential actions and states (situations) are numerous or intricate. DQN tackles this by integrating deep learning, a form of advanced machine learning. It uses deep neural networks, which are sophisticated models capable of handling vast and complex data, to estimate the best action to take in each state [8] [9].

The use of DQN in the context of managing a highly dynamic, geo-distributed cloud platform, particularly for data resource allocation, is well-suited due to the complexity and high dimensionality of the action space in such environments. Traditional machine learning and optimization methods might struggle in efficiently navigating this space to make effective resource allocation decisions. This complexity necessitates an approach like DQN, which can effectively handle such high-dimensional spaces.

In DQN, the policy function $\pi : S(t_s) \times A(t_s) \rightarrow [0, 1]$ maps the state-action pairs to the probability of choosing an action $A(t_s)$ in a given state $S(t_s)$. The action-state function $Q(S(t_s), A(t_s))$ quantifies the effectiveness of each action in a state. The Q-function is expressed as the expected sum of discounted future rewards:

$$Q(s, a) = \mathbb{E}\left[\sum_{k=1}^{N} \gamma^k R(t_s + k) \mid S(t_s) = s, A(t_s) = a\right] \quad (4)$$

where $\gamma \in [0, 1]$ is the discount factor, balancing the immediate and long-term rewards. In traditional Q-learning, a Q-table is updated for each state-action pair, but in high-dimensional spaces, this becomes impractical due to the exponential growth of the state-action space.

To overcome this, DQN uses deep neural networks to approximate the Q-function, denoted as $Q(s, r, \theta) \approx Q(s, r)$, with θ representing the network weights. The network is trained to minimize the loss function:

$$L(\theta) = \mathbb{E}\left[(R(t_s) + \gamma \max_{a'} Q(s', a', \theta') - Q(s, a, \theta))^2\right] \quad (5)$$

This approach enables the agent to learn an effective policy for resource allocation in the complex, high-dimensional environment of a cloud platform without needing a large Q-table. The detailed algorithm for DQN is Algorithm 1 shown in Appendix A.

3.2 Double DQN

The Double Deep Q-Network (DDQN) algorithm is an enhancement over the standard Deep Q-Network (DQN), designed to address a specific shortcoming in DQN known as the overestimation bias. This bias arises from the use of the same network to both select and evaluate actions, leading to overly optimistic value estimates. In DDQN, two neural networks are employed: the principal network for action selection and the target network for action evaluation. This separation helps in reducing the overestimation bias seen in standard DQN. The DDQN algorithm updates the Q-values as follows:

$$Q(s, a) \leftarrow Q(s, a) + \alpha\left[T - Q(s, a, \theta)\right] \quad (6)$$

$$T = R(s, a) + \gamma \max_{a'} Q_{\text{target}}(s', a', \theta') \quad (7)$$

where $R(s, a)$ is the immediate reward, γ is the discount factor, and $Q_{\text{target}}(s', a')$ represents the maximum Q-value for the next state s' as evaluated by the target network.

Here, s and s' represent the current and next state, respectively, a is the action taken, $R(s, a)$ is the reward received, α is the learning rate, and γ is the discount factor. $Q_{\text{principal}}$ and Q_{target} are the principal and target networks, respectively.

3.3 Count-based DQN

Instead of using epsilon-greedy, we have implemented a count-based optimistic exploration algorithm to improve the selection of actions. The state space comprises four dimensions, with one having an infinite range (predicted viewers) and two being continuous (cost, IAD), posing a challenge for conventional optimistic exploration methods, since most states occur only once. To address this, we used a simple hash code mapping for the first two dimensions with Python hash methods. Our first attempt compressed the second dimension - predicted viewers into a binary variable,

and combined with the data center index to represent the states. In the future, this approach can be extended by incorporating other locality-sensitive hashing techniques, spreading the four dimensions, including SimHash, k-mean neighbors, autoencoder, or more recent vector quantization methods, such as residual vector quantization, finite scalar quantization or lookup free quantization. After the compression, the algorithm proposed by Tang et. al. [10] updates the policy π using rewards:

$$R(s,a) \leftarrow R(s,a) + \frac{\beta}{\sqrt{n(\phi(s))}} \tag{8}$$

where ϕ is a hash function. However, instead of updating $R(s,a)$, we update $Q(s,a)$ to keep the cumulative rewards comparable to other methods. A snippet of the algorithm is Algorithm 3 in Appendix A. A large portion of the base DQN algorithm is maintained, including the implementation of buffer and target network. We replace the epsilon-greedy exploration-exploitation with a greedy argmax operation. We update Q-target more slowly for states that are visited more frequent and vice-versa. Note that since the state-action space is relatively small, we use a state-space count $n(s)$ rather than a state-action count $n(s,a)$.

4 Result

We divided the streams into 90% entries for the training task and 10% for the evaluation. The rental costs, migration costs, and serving costs follow the charging model EC2 and S3 of Amazon. We assumed all videos have the same size of 0.738 GBit. The round-trip delay matrix is created by finding the ping times between different geo-distributed cloud sites from WonderNetwork. We consider the latency threshold of 220 ms. To compare the performance of the reinforcement learning algorithms in this study, we use the same architecture and hyperparameters for all the neural networks. The deep learning architecture employed consists of two linear layers, both with 128 units and ReLU activation. Details of the hyperparameters are shown in Appendix B.

A comparative analysis of the performance of four algorithms based on the moving average of training rewards in the first 15,000 steps is illustrated in **Fig.1**. The greedy algorithm serves as the baseline, representing an allocation of resources to the data center with the largest number of predicted viewers, determined by the predictive machine learning model. The Benchmark Greedy algorithm exhibits a steady performance, oscillating around a reward value of 5. The greedy allocation results in such pattern and does not have any complicated exploration and learning process. The performance of the base DQN and Double DQN algorithms are closely aligned throughout the training period, with their curves almost overlapping and converges to about 5.5 at around timestep equal to 2500. This indicates that the additional complexity of the Double DQN, which is designed to mitigate the overestimation of action values, does not yield a noticeable benefit over the standard DQN in this context. Both algorithms show a steady increment in rewards. The Optimistic Count-based DQN stands out, showing not only a rapid ascent in the reward average and faster convergence, indicating a swift learning curve, but also achieving the highest performance levels though still at about 5.5. This suggests that the Optimistic Count-based DQN is effective at quickly assimilating information and utilizing it to make profitable decisions in the environment. Note that the training rewards of our RL algorithms reach an approximate value of 5.5, whereas the rewards for the greedy approach remain consistently around 5. This suggests that RL, throughout continuous learning, is capable of capturing the dynamics of crowdsourcing system, enabling it to adapt and improve its performance over time.

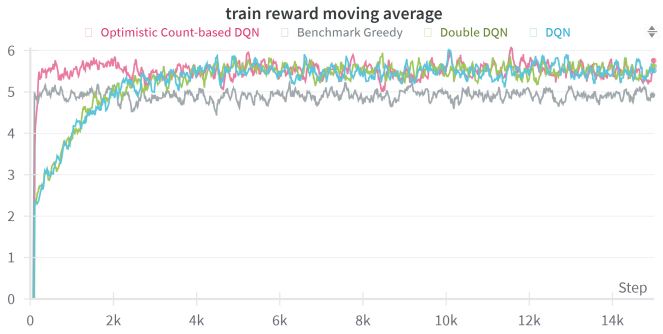

Fig. 1. Training Reward Moving Average for 15000 timesteps

Fig.2 presents the moving average of evaluation rewards for four distinct algorithms. The Greedy algorithm, with its strategy of a singular allocation based on peak predicted viewership, exhibits considerable variability in performance, oscillating between just above 5.2 and near 4.5. Reinforcement learning algorithms, in general, outperform the Greedy algorithm in terms of median reward. Surprisingly, Double DQN demonstrates a broader fluctuation in rewards, with its peaks surpassing 5.8 and occasionally reaching as high as 5.9. This contrasts with the base DQN, whose performance consistently remains below the 5.8 mark. On the other hand, Optimistic Count-based DQN not only reaches above the 5.8 threshold but also shows a more reliable and stable pattern when compared to Double DQN. Throughout the evaluation, all three reinforcement learning algorithms demonstrate a notable steadiness in their average rewards, suggesting a robust performance over the course of training. Though there is no strong evidence to support the differences between three algorithms, we suppose that optimistic count-based DQN gives a relatively optimistic choice in practice given its consistency and high evaluation reward.

5 Conclusion

This study explores the challenge of delivering cost-effective crowdsourced live streaming services on a geo-distributed cloud platform with a primary focus on enhancing viewers' Quality of Experience (QoE). Using a predictive machine learning model, we estimated the potential number of viewers at each cloud site based on content features. We formulated a RL problem to maximize viewer satisfaction while minimizing costs and meeting a predefined network delay threshold. We explored three variations of Deep Q Networks (DQN): the base DQN, double DQN, and count-based optimistic DQN. Our findings illustrate that our RL

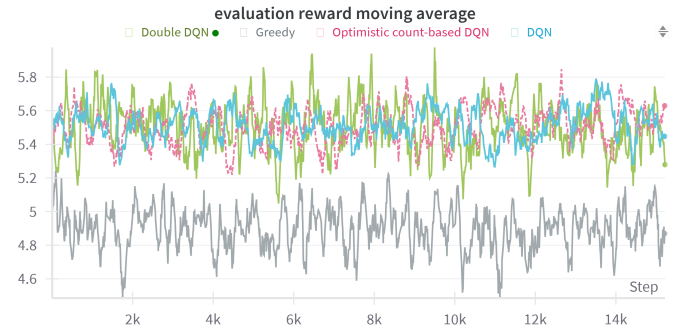

Fig. 2. Evaluation Reward Moving Average

approach outperforms the baseline greedy approach. Additionally, we also figure out that for this specific context and with the large dataset, the base DQN and double DQN don't have obvious differences on their performance on final convergence values and converging rates. But count-based optimistic DQN gives a pretty high converging rate, which may be very useful for small-scale dataset. For example, for specific categories that don't have many samples, count-based DQN may give a much better result in practice than the base DQN and double DQN.

In the future, we would like to explore different architectures for DQNs to enhance the performance of our models. Our focus will extend beyond the conventional structures, aiming to modify the rate of convergence and develop more robust designs. We plan to investigate the efficacy of multiple Q-networks employing specialized selection methods and criteria, such as random choice. Additionally, we will explore the potential benefits of employing attention mechanisms to combine various Q-networks and determine optimal actions for each time step. Combining UCB (Upper Confidence Bound) selection with Q-network is another avenue we intend to explore for action determination. Data improvement will be a crucial aspect of our future work, involving the use of different networks tailored to specific video characteristics, such as size and broadcaster location. We also plan to employ specialized reinforcement learning techniques tailored to different categories of videos, fostering a more adaptive and nuanced approach to optimization. The specialized reinforcement learning may have a more reasonable application in practice, and the total reward should be higher. For more complex settings with higher dimensional state space, we also want to see the performance of more advanced hashing methods to incorporate the optimistic DQN algorithm. These pursuits collectively aim to contribute to the evolution and refinement of our models in the dynamic landscape of Deep Q Networks. Finally, we would like to compare the performance of our RL algorithm with IPL heuristic optimization approaches.

APPENDIX A
ALGORITHMS PSEUDOCODES

The DQN learning process can be summarized into several key steps as follows:

- **Initialization (line 1):** The algorithm begins by setting the hyper-parameters such as learning rate, batch size, replay buffer D, and the rates for exploration and exploitation. The environment is prepared with the video data V, and the state and action sizes are defined. The Q-function networks, both the principal Q and the target Q_{target}, are initialized.
- **DQN Learning Setting (line 4-5):** The learning loop starts for each video v_i in the dataset V. Variables for the Incremental Average Delay (IAD) and deferred reward (dr) are set to zero, and the initial migration cost is established.
- **Time-Step Iteration (line 7-8):** At each time-step t_s for video v_i, the delay is initialized, and the current state information is retrieved, including the data center r_{t_s}, predicted viewers $p(v_i, t_s, r_{t_s})$, cost, and IAD.
- **Action Selection (line 9-13:** The algorithm decides to either explore by choosing a random data center or exploit by selecting the data center with the highest Q-value based on the current policy.
- **Step Function - Calculating Cost, Delay, and Reward (line 14-31):** The cost is updated for each data center if necessary, the delay for the current time-step is calculated, and the reward is set based on the change in cost. The IAD is updated, and the reward is adjusted if the IAD is below a certain threshold.
- **Buffer Update and Network Training (line 32-38):** After observing the reward $R(t_s)$ and the next state $S(t_s + 1)$, the experience tuple is stored in the buffer D. When conditions are met, a mini-batch is sampled from the buffer to train the Q-Principal network.
- **Network Update (line 39-41):** The target Q-network is periodically updated with the weights from the principal Q-network at specified intervals defined by the update step τ_{update}.

The major difference between Double DQN and DQN is that, instead of using the maximum Q-value from the target network for the next state (as in DQN), DDQN evaluates the action selected by the principal Q network using the target Q network. This decouples the selection and evaluation of the best action and is intended to reduce overestimations.After sampling from the buffer, for each sample in the mini-batch, the action a' for the next state $S(j + 1)$ is chosen using the principal Q-network (Q-Principal). This action a' is then evaluated using the target Q-network (Q-Target). The target Q-value for training is calculated using the reward $R(j)$ and the discounted Q-value of the action a' evaluated by the target network. The principal Q-network is trained using these targets. The target network is updated periodically as per the update frequency τ_{update}.

In the optimistic count-based DQN algorithm, the state processor ϕ is crucial for handling the infinite and continuous dimensions of the state space. By hashing the state space, we can efficiently implement a count-based method

Algorithm 1 DQN Learning Process

Initialization:

1) Hyper-parameters: learning rate, batch size, buffer D, exploration-exploitation rate, decay rate, etc.
2) Initialize environment objects: video data V, environment, state size, action size (M+1)
3) Q-function networks (principal Q and target Q_{target})

DQN Learning:
for each episode $v_i \in V$ **do**
  $IAD = 0, dr = 0$
  $Cost(0)$ = initial migration cost
  for each time-step $t_s = 1, \ldots, t_N^{v_i}$ **do**
    Initialize delay $delay_{t_s} = 0$
    Get Current Info from State: data center r_{t_s}, predicted number $p(v_i, t_s, r_{t_s})$, $Cost$, IAD= $S(t_s)$
    if Explore **then**
      Select Random data center $A(t_s)$ = random (0, M)
    else Exploit
      Select data center based on Q-value $A(t_s) = \arg\max Q(S(t_s), A(t_s))$
    end if
    for each data center$k = 1 \ldots M$ **do**
      if $r_{t_s} \neq$ broadcaster's data center $r_b^{v_i}$ **then**
        Update Cost
      end if
      Update delay $delay_{t_s}$
    end for
    Update IAD $IAD = IAD + delay_{t_s}$
    Set Reward $R(t_s) = Cost(t_s) - Cost(t_s - 1)$
    if $(\sum A(t_s) = 1 \wedge p(v_i, t_s, r_{t_s}) > 0) \vee (\sum A(t_s) = 0 \wedge p(v_i, t_s, r_{t_s}) = 0)$ **then**
      if $IAD < D$ **then**
        $R(t_s) = R(t_s) + 1$
      else
        $dr = dr + 1$
      end if
    end if
    if $IAD < D \wedge t_s = t_N^{vi}$ **then**
      $R(t_s) = R(t_s) + dr$
    end if
    Observe $R(t_s)$ and the next state $S(t_s + 1)$.
    Save $(S(t_s), A(t_s), R(t_s), S(t_s + 1))$ in buffer D.
    if (total step $>$ initialize step) **and** (total step $\mod \tau_{train} = 0$) **then**
      Sample a mini-batch of $(S(j), A(j), R(j), S(j + 1))$ from D.
      Find target Q-value $Q_{target}(j)$ from target Q-network $Q_{target}(j) = R(j) + \gamma \times \max_{a'} Q(s', a', \theta')$.
      Train Q-Principal Q
    end if
    if (total step $\%$ τ_{update}) $== 0$ **then**
      Update the target Q-network
    end if
  end for
end for

Algorithm 2 Double DQN

Initialization: ▷ Same as DQN
DDQN Learning: ▷ Start of modifications for DDQN
for each episode $v_i \in V$ **do** ▷ Same as DQN
  for each time-step $t_s = 1, \ldots, t_N^{v_i}$ **do** ▷ Same as DQN
    Initialize, Take Action, and Step ▷ Same as DQN
    Observe $R(t_s)$ and the next state $S(t_s + 1)$.
    Save $(S(t_s), A(t_s), R(t_s), S(t_s + 1))$ in buffer D.
    if (total step $>$ initialize step) **and** (total step mod $\tau_{train} = 0$) **then**
      Sample a mini-batch from D.
      for each sample in the mini-batch **do**
        Select action a' for $S(j + 1)$ using Q-Principal: $a' = \arg\max_a Q(S(j + 1), a, \theta)$
        Evaluate a' using Q-Target: $Q_{eval}(j) = Q(S(j + 1), a', \theta')$
        Calculate target: $Q_{target}(j) = R(j) + \gamma \times Q_{eval}(j)$
      end for
      Train Q-Principal Q
    end if
    if (total step $\%\tau_{update}$) $== 0$ **then**
      Update the target Q-network
    end if
  end for
end for

to encourage exploration of less frequently visited states. This method updates the Q-values by incorporating an exploration bonus inversely proportional to the square root of the state visitation count, thereby promoting exploration in a more balanced manner compared to epsilon-greedy strategies.

The full algorithms pseudo-codes for Double DQN and Optimistic count-based DQN are not provided due to the volume limit and to avoid redundancy from repetition.

Algorithm 3 Snippet of Count-based Exploration with Static Hashing

Define state processor $\phi : S \to R^D$
Initialize $n(.) \leftarrow 0$
Initialize a hash table with values $n(\cdot) \equiv 0$
for each iteration k **do**
  Take action $a \leftarrow \text{argmax}_{a'} Q^k(s, a')$
  Compute hash codes through any LSH method $\phi(s)$
  Update the hash table $n(\phi(s)) \leftarrow n(\phi(s)) + 1$
  $Q^{k+1}(s, a) \leftarrow r + \gamma \max_{a'} Q^k(s', a') + \frac{\beta}{\sqrt{n(\phi(s))}}$
  Continue with the DQN.
end for

APPENDIX B
TABLE OF NOTATION

TABLE 1
Table of notations

Notation	Description
M	Number of geo-distributed cloud sites.
R^2	Determination coefficient(,Machine Learning Metrics).
$\bar{v}$	Mean number of viewers among all records.
V	Total number of videos.
v_i	Video index.
Z_v	Size of a video.
r_j	Region index.
$r_b^{v_i}$	Broadcasting site of video v_i.
$r_a^{v_i}$	Allocation region of video v_i.
$r_s^{v_i}$	Viewers' region where v_i is served.
d	delay.
c_{r_j}	Rental cost per GB in the region r_j.
$H(v_i, r_a^{v_i})$	Decision variable, indicates that v_i is hosted in the site $r_a^{v_i}$.
$p(v_i, r_i)$	Predicted number of viewers in the streaming region r_i.
$Cost$	Total operational cost.
c_j^r	Rental/Server cost.
$(c_{r_a}^{r_b})$	Migration/Switch cost from r_a to r_b
(c_j^s)	Serving/Service cost.
D	Delay threshold.
(S, A, R)	State, Action, Reward.
t_s	Time-step.
IAD	Incremental Average Delay.
dr	Deferred reward.
π	RL policy function.
Q	Action-state function.
D	RL Buffer.
γ, α	Discount factor, learning rate.
Q	principal Q-network.
θ	Weights of the deep network.
L	Loss function.
Q_{target}	target Q-network.

APPENDIX C
HYPERPARAMETERS TABLES

TABLE 2
Hyperparameters of MLP, Random Forest, and XGBoost

Parameter	MLP	Random Forest	XGBoost
Input size	Variable	N/A	N/A
Output size	Variable	N/A	N/A
Number of layers	5	N/A	N/A
Neurons per layer	128 to 1024	N/A	N/A
Activation	ReLU	N/A	N/A
Number of estimators	N/A	100	100
Max depth	N/A	Unlimited	6
Learning rate	N/A	N/A	0.3
Subsample	N/A	N/A	1
Col sample by tree	N/A	N/A	1

TABLE 3
Hyperparameters of Reinforcement Learning

Parameter	Value
Learning Rate (lr)	5×10^{-4}
Batch Size	64
Batch Size (Optimistic Count-DQN)	8
Replay Buffer Size (maxlength)	1000
Target Network Update Frequency (tau)	100
Initial Steps Before Training (initialsize)	500
Exploration Start (EPS_START)	1.0
Exploration End (EPS_END)	0.0001
Exploration Decay Units(EPS_DECAY)	10000

TABLE 4
Hyperparameters of the DQN Model

Parameter	DQN Network
Input size	`obssize`
Output size	`actsize`
Number of hidden layers	2
Neurons in first hidden layer	128
Neurons in second hidden layer	128
Activation function	ReLU

ACKNOWLEDGMENTS

The authors extend their sincere gratitude to Professor Shipra Agrawal for her invaluable guidance and insightful instructions on the project, as well as for her dedication and support throughout the semester. Special thanks are also due to teaching assistants Chenyu Zhang and Poch Laohrenu, whose expertise and assistance were instrumental in the success of this project and the enrichment of the course. The code for the algorithms discussed in this paper is available at https://drive.google.com/drive/folders/1brSW88qSjCUylIcvzCPSPO7n0r5E7Ith?usp=sharing with restriction to Lionmail users.

REFERENCES

[1] K. Bilal, A. Erbad, and M. Hefeeda, "Qoe-aware distributed cloud-based live streaming of multisourced multiview videos," *Journal of Network and Computer Applications*, vol. 120, pp. 130–144, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1084804518302364

[2] C. Dong, W. Wen, T. Xu, and X. Yang, "Joint optimization of data-center selection and video-streaming distribution for crowdsourced live streaming in a geo-distributed cloud platform," *IEEE Transactions on Network and Service Management*, vol. 16, no. 2, pp. 729–742, 2019.

[3] F. Chen, C. Zhang, F. Wang, J. Liu, X. Wang, and Y. Liu, "Cloud-assisted live streaming for crowdsourced multimedia content," *IEEE Transactions on Multimedia*, vol. 17, no. 9, pp. 1471–1483, 2015.

[4] F. Haouari, E. Baccour, A. Erbad, A. Mohamed, and M. Guizani, "Transcoding resources forecasting and reservation for crowdsourced live streaming," in *2019 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2019, pp. 1–7.

[5] X. Yuan, M. Sun, Q. Fang, and C. Du, "Dlecp: a dynamic learning-based edge cloud placement framework for mobile cloud computing," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2019, pp. 1035–1036.

[6] E. Baccour, A. Erbad, A. Mohamed, F. Haouari, M. Guizani, and M. Hamdi, "Rl-opra: Reinforcement learning for online and proactive resource allocation of crowdsourced live videos," *Future Generation Computer Systems*, vol. 112, pp. 982–995, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X20306269

[7] E. Baccour, A. Erbad, K. Bilal, A. Mohamed, M. Guizani, and M. Hamdi, "Facebookvideolive18: A live video streaming dataset for streams metadata and online viewers locations," in *2020 IEEE International Conference on Informatics, IoT, and Enabling Technologies (ICIoT)*. IEEE, 2020, pp. 476–483.

[8] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013. [Online]. Available: https://arxiv.org/pdf/1312.5602.pdf

[9] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[10] H. Tang, R. Houthooft, D. Foote, A. Stooke, X. Chen, Y. Duan, J. Schulman, F. D. Turck, and P. Abbeel, "#exploration: A study of count-based exploration for deep reinforcement learning," *CoRR*, vol. abs/1611.04717, 2016. [Online]. Available: http://arxiv.org/abs/1611.04717