*Regular Article*

# Deep Reinforcement Learning - based Bitrate Adaptations in Dynamic Adaptive Streaming over HTTP

**Long Minh Luu[1,2], Nghia Trung Nguyen[1,2], Phuong Luu Vo[1,2], Tuan-Anh Le[3]**

[1] International University, Ho Chi Minh City, Vietnam
[2] Vietnam National University, Ho Chi Minh City, Vietnam
[3] Thu Dau Mot University, Binh Duong, Vietnam

Correspondence: Phuong Luu Vo, vtlphuong@hcmiu.edu.vn
Communication: received 18 February 2022, revised 04 October 2022, accepted 06 October 2022
Online publication: 10 October 2022, Digital Object Identifier: 10.21553/rev-jec.308
The associate editor coordinating the review of this article and recommending it for publication was Prof. Vo Nguyen Quoc Bao.

*Abstract*– **Dynamic adaptive streaming over HTTP (DASH) has been a superior video streaming technology in recent years. Bitrate adaptation function at video player plays a vital role in guaranteeing a high quality-of-experience for the users. This work evaluates the performance of several advanced deep reinforcement learning algorithms, *i.e.*, deep Q-learning, actor-critic, and proximal policy optimization, applied in bitrate adaptations and compares them with other rate adaptation methods with real-trace datasets.**

*Keywords*– **Dynamic adaptive streaming over http, bitrate adaptation, deep reinforcement learning.**

## 1 Introduction

A significant part of Internet traffic today is video streaming [1], in which dynamic adaptive streaming over HTTP (DASH) [2] is an effective technique to stream a video from the server to the client. A DASH's video is chunked into multiple chunks with equal playback time and encoded at different levels of quality. A higher quality chunk results in a larger chunk size. The video player requests video chunks one by one and plays them in order. The chunk is not played until it is fully downloaded.

With the variation of network conditions, the rate adaptation function at the video player has an essential role in improving the user's quality of experience (QoE). The QoE metrics include the quality of the video, the smoothness of the video chunks (small number of quality fluctuations), and no video stalling. The adaptation method chooses a quality level for each chunk to request based on the current conditions, *e.g.*, network speed, current buffer size, *etc.*

Despite the simplicity of DASH framework, there are several challenges a rate adaptation algorithm must address [3, 4]:

1) Varying network conditions: the network speed depends on factors such as signal strength, the number of users, time of day, *etc.* fluctuate frequently; the video chunk sizes are different even within a quality level, *etc.*
2) Multi-purpose balance: a rate adaptation algorithm must balance between 1) maximizing video quality, 2) minimizing rebuffering time, and 3) maintaining video smoothness. These problems are usually in conflict with each other. For example, for a low-speed network, maximizing video quality will increase the rebuffering time.
3) Long time horizon: a good rate adaptation algorithm does not simply select the chunk which yields the highest QoE for the subsequent request, but all the upcoming chunks to maximize the QoE of a whole episode. For example, choosing high-quality chunks will increase the user's QoE; however, it has a long download delay and may cause video rebuffering.

Various adaptation algorithms have been proposed to improve rate adaptation algorithms. Throughput-based adaptation algorithms choose the quality level for the next chunk based on the estimated throughput from downloading the previous chunks [2]. The buffer-based algorithms observe the buffer level to decide the quality level [5]. Some methods combine these approaches, such as [6].

The success of deep Q-learning (DQN), which achieved human-level playing in the Atari game [7] has inspired the development of various deep reinforcement learning (DRL) algorithms such as asynchronous advantage actor-critic (A3C) [8], trust-region policy optimization (TRPO) [9], proximal policy optimization (PPO) [10], *etc.* and the applications of DRL algorithms in many fields. Recently, the works [11, 12] have applied DRL algorithms to improve the performance of rate adaptation in DASH. D-DASH in [12] uses deep Q-learning (DQN), and Pensieve [11] uses asynchronous advantage actor-critic (A3C). The state is a combination of several observations, *e.g.*, estimated throughput of the previous chunk, chunk sizes, remaining chunks, delay, *etc.* Neuron networks map the state to Q-values (DQN) or actions (A3C) in DRL algorithms.

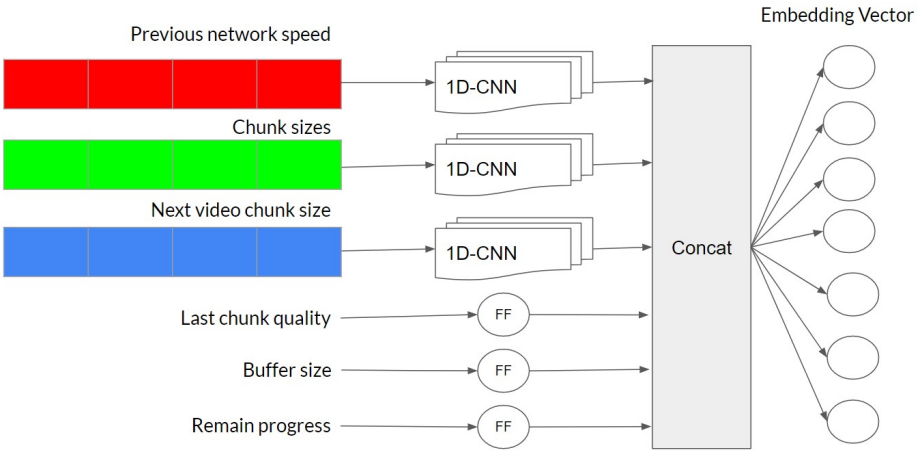



Figure 1. Features extractor network.

Both DRL-based methods achieve a higher QoE than the traditional adaptation methods. However, DQN and A3C are also known for their sensitivity to the variation of the hyperparameters.

In this paper, we compare the performance of different DRL algorithms to enhance the performance of DASH. DQN, actor-critic, and state-of-the-art PPO algorithms are used in the framework. We build a simulation environment to train and evaluate the models. It allows the agent to experience ten thousand video episodes within hours. The real-world bandwidth datasets from 4G LTE and Federal Communication Commission (FCC) are used. To the best of our knowledge, this is the first work comparing the performance of DASH with various DRL algorithms.

The remaining paper has the following structure. Section 2 presents the DRL framework applying to DASH's rate adaptation. The simulation environment, datasets, and other environment setups are described in Section 3. Sections 4 and 5 show the results and conclude the work, respectively.

## 2 DRL-based Adaptive Video Streaming Model

Assume that the video has $N$ chunks and $K$ quality levels. In the DRL-based rate adaptation, step $n$ is defined as an event that the agent takes action downloading chunk $n$. An episode includes $N$ steps. Let $B_n$ be the bitrate of the quality level of chunk index $n$.

### 2.1 Action Space and State Space

**Action space**: *Action space* has $K$ actions which corresponding to the number of quality levels of the video.

**State space**: At the beginning of step $n$, a *state* is observed from the environment, which includes:

- A vector of the estimated network speed $\{C_{n-1}, \ldots, C_{n-6}\}$ in Mbps.
- A vector of the chunk's size $\{\sigma_{(n,1)}, \ldots, \sigma_{(n,K)}\}$ in Mbps.
- A vector of previous download time $\{d_{n-1}, \ldots, d_{n-6}\}$ in seconds.
- A scalar value indicates current buffer size $G_n$, normalized by 10.
- A scalar value indicates the number of remaining chunks, normalized by the episode's length.
- A categorical value indicates the last chunk quality level.

Similar to [11], we also use a simple network to extract features of a state into an embedding vector, which is described in Figure 1. Particularly,

- The scalars are passed through feedforward layers of 128 units with ReLU activation function.
- The arrays are passed through 1D-CNN layers of 128 filters with kernel size 4 and stride 1, followed by ReLU.
- Except for the last chunk quality: it is encoded to a one-hot vector and then is passed through a feedforward layer of 128 units, followed by ReLU.

These output values are flattened, concatenated, and then passed to two layers with Tanh activation function in between. The output of the features extractor is an embedding vector, which is the input of the policy and the value networks. The feature extractor is shared between the actor and value networks to accelerate computing.

### 2.2 Reward Function

QoE is used as the reward of DASH model as in [5, 11, 12]. It combines the bitrate utility, the quality switch penalty, and the rebuffering penalty.

Let $q(B_n)$ be the function indicating the utility downloading chunk $n$ at bitrate $B_n$. Following [5], we use logarithm $q(B_n) = \ln(B_n/B_{min})$ for utility function. The logarithm scale captures the notion that the marginal improvement in perceived quality decreases at the higher bitrates.

Let $\phi_n$ be the rebuffering time that results from downloading chunk $n$ at bitrate $B_n$. Assume that $G_n$ is the buffer size at the beginning of step $n$ and $d_n$ is the download time of chunk $n$. If $d_n > G_n$, we have

$\phi_n = d_n - G_n$. Otherwise, the video have no rebuffering when playing chunk $n$. Therefore, rebuffering time corresponding to chunk $n$ is calculated by the following formula:

$$\phi_n = \max(0, d_n - G_n).$$

The *reward* corresponding to chunk $n$ is given by

$$r_n = q(B_n) - \mu_s|q(B_n) - q(B_{n-1})| - \mu_\phi \phi_n, \quad (1)$$

where the term $\mu_s|q(B_n) - q(B_{n-1})|$ penalties the switching in quality levels of two consecutive chunks with weight $\mu_s$.

## 3 Experiment Details

### 3.1 Environment

We build an event-driven simulation to train the DRL agent. A simulated environment consists of two main functions: *step* and *reset*. The *reset* function is called when starting an episode to reset the environment to the beginning of an episode. After that, *step* function is called consecutively until the episode terminates. The *step* function requires action as input and returns a new state, reward, and a boolean value indicating if the episode is terminated.

In DASH, the downloaded chunks are stored in the client buffer before being played. Buffer size is measured by the total playing time of the wait-to-be-played chunks. The buffer follows the first-in-first-out rule. When the buffer exceeds an upper threshold level $G^{\max}$, the video player will stop requesting new chunks. It must wait for the buffer level to decrease below $G^{\max}$ to send the request for a new chunk again. The video player *re-buffers* when the chunk index going to be played does not exist in the buffer. The video player pauses playing and waits for the new chunk to be completely downloaded. Re-buffering causes the video to freeze.

To emulate the process of downloading and playing video chunks in DASH, we use a list of events that includes five types, *i.e.*, DOWN, DOWNF, SLEEP, PLAY, and REBUFFER. Every event is associated with a timestamp. Events are sorted in the order of timestamps. Each event encountered generates a new one.

**DOWN, DOWNF, and SLEEP events**: A DOWN event is generated when the video player requests a new chunk, a DOWNF event is generated when a chunk is completely downloaded, and a SLEEP event is generated when the player pauses requesting a new chunk when the current buffer size is higher than the threshold $G^{\max}$.

- When a DOWN event requesting chunk $n$ with timestamp $t$ is encountered, a DOWNF event is generated at time $t + d_n$, in which $d_n$ is the chunk's download time.
- When a DOWNF event with timestamp $t$ is encountered, a DOWN event associated with the next chunk is appended to the event list if the buffer size is less than $G^{\max}$; otherwise, a new SLEEP event is generated at time $t + sample$, where $sample = 0.02$. Let $G$ be the current buffer size when a DOWN event associated with chunk $n$ is encountered (requesting chunk $n$). The buffer size is updated at DOWNF event by the formula:

$$G := \max(0, G - d_n) + T, \quad (2)$$

where $d_n$ is the download time of chunk $n$, and $T$ is the playback time of chunk.
- When a SLEEP event with timestamp $t$ is encountered, a new DOWN event associated with the next chunk is generated with timestamp $t + sample$ if $G < G^{\max}$; otherwise, a new SLEEP event is appended. The buffer size is updated at the SLEEP event as follows:

$$G := G - sample. \quad (3)$$

**PLAY and REBUFFER events**: A PLAY event is generated when the player starts playing a chunk, and a REBUFFER event is generated when the chunk intended to play is not in the buffer.

- When a PLAY event playing chunk $n$ with timestamp $t$ is encountered, a new PLAY event associated with chunk $n + 1$ with timestamp $t + T$ is generated if chunk $n + 1$ is completely downloaded. Otherwise, a REBUFFER event with timestamp $t + T + sample$ is generated.
- When a REBUFFER event with timestamp $t$ is encountered, a new PLAY event associated with the next chunk index is appended to the event list if the next-playing chunk index is completely downloaded. Otherwise, a new REBUFFER event with timestamp $t + sample$ is appended.

An episode is initialized by *reset* function with a DOWN event and a REBUFFER event at timestamp 0. A *step* of the environment is the period between two consecutive DOWN events. The episode terminates after the last chunk of the video player is played.

### 3.2 Datasets

**Video and quality levels**: We use the Elephant Dream video dataset [13] encoded into 20 different quality levels for each 4-second chunk. We choose 07 bitrate levels: $B_n = (700, 900, 2000, 3000, 5000, 6000, 8000)$ Kbps, following the guidelines by [14], which map to approximately the following human-friendly quality levels: (240p, 360p, 480p, 720p, 720p@60fps, 1080p, 1080p@60fps). Thus, the DRL agent has seven discrete actions in each step. We use the first 60 chunks of the video for each episode which is 240 seconds in training and evaluation. The default quality at the beginning of each episode is the lowest quality level.

**4G LTE**: the 4G LTE dataset [15] contains 135 traces, with an average duration of 15 minutes per trace, at 1-second granularity. This dataset collected traces from Irish mobile operators with five mobility patterns: static, pedestrian, car, bus, and train. By using 60 seconds sliding window across this dataset, we also generated 1,000 traces with 320 seconds per trace.

**FCC**: The FCC dataset contains over 1 million throughput traces, at a granularity of 10 samples per

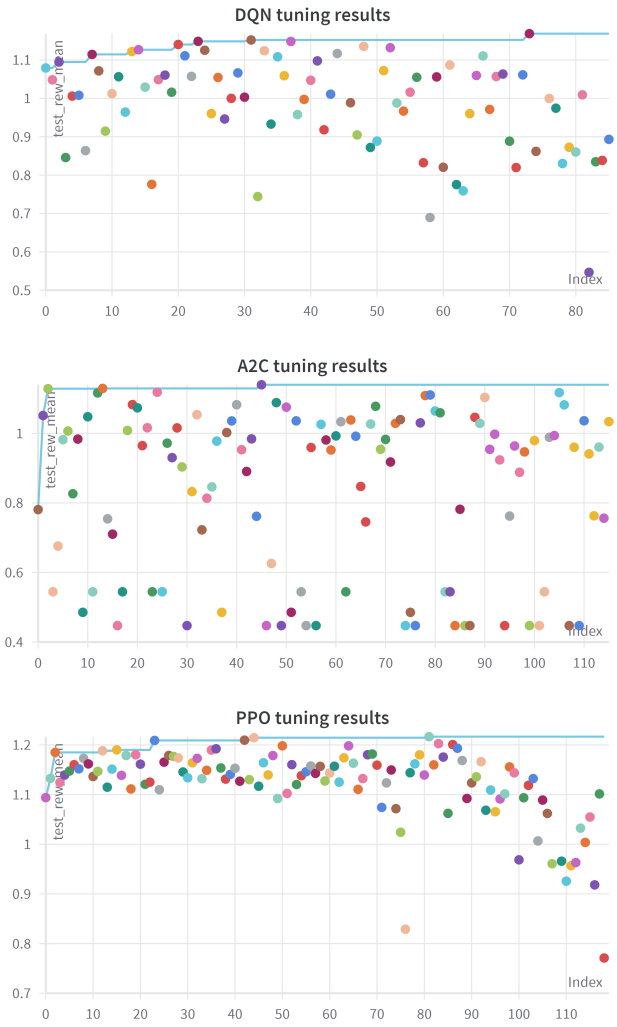



Figure 2. Tuning results of DQN, A2C, and PPO. Y-axis is mean reward and X-axis is the number of trails, which is 120.

second [16]. We construct 1,000 random traces (each spanning 320 seconds) for our dataset. Although the method of collecting data is very similar to [11, 12], our throughput traces are randomly selected from the "download speed" category instead of using "web browsing" as in [11]. According to FCC technical report [16], throughput in "web browsing" is measured by taking the sum of sizes of all the contents of the web page divided by the fetch time, which is the time consumed to download those contents. However, because the size of web pages varies, the fetch time also varies. In "download speed" category, the measurement time is 10 seconds. The client attempts to download as much as possible, and the average throughput achieved during this period is recorded. In this way, the network segment will ensure granularity in 10 seconds. We use the September 2019 collection.

In each FCC or LTE dataset, we randomly split 80% of the dataset to train and 20% to test, which are 800 and 200 traces, respectively. At the beginning of every episode, the trace and the initial point are picked randomly from the train set. We concatenate the FCC and LTE training sets to train the DRL agents.

### 3.3 Hyperparamter Tunning

We apply three popular DRL algorithms, *i.e.*, deep Q-learning (DQN), advantage actor-critic (A2C), and proximal policy optimization (PPO). To find a good hyperparameter set, each algorithm is run for $\approx 120$ trials with different seeds. In each run, hyperparameters are uniformly selected from ranges of hyperparameters. The best hyperparameter set in 120 trials is used to train and test again for the final results with ten different seeds. More information about the hyperparameter tuning strategy can be found in the appendix.

The rewards of 120 trials of DQN, A2C, and PPO are shown in Figures 2. We can see that *PPO is more stable than A2C and DQN when varying hyperparameters*.

### 3.4 Training and Testing

**Training library.** As noted by [17–20], code-level optimization tricks (*e.g.* observation normalization, reward scaling, global gradient clipping) from different codebases greatly influence the performance of DRL algorithms. Thus, we use the implemented algorithms in Stable-Baselines3 [21], a well-known and reliable DRL codebase, to train the model. PPO and A2C are trained with four parallel environments; however, parallel environments are not supported with DQN. For PPO and DQN, Adam optimizer is used according to [22]. For A2C, we add the option of using RMSprop [23] or Adam in tuning this hyperparameter. Each algorithm trains in $885,000$ steps, which is around $15,000$ episodes. The readers can find the open-source codes for training and tuning at [24] and [25].

## 4 Results

We compare the DRL-based rate adaptation methods with some other non-DRL methods, *i.e.*, random, constant, throughput-based [2] and BOLA [5]. Particularly,

- Random: the next quality is picked randomly at each step.
- Constant: at each step, the next quality is picked as the 3000Kbps quality (which is known as HD (720p) quality).
- Throughput-based (THRB): at each step, the download chunk has the highest quality level, of which the corresponding bitrate is smaller than the harmonic mean of the estimated throughputs of three previous downloaded chunks.
- BOLA: the buffer-based adaptation method that uses Lyapunov optimization to minimize rebuffering and maximize video quality.

### 4.1 Training Curve and Testing QoE Metrics

We first present the results with the best hyperparameters in Table I with case $\mu_\phi = 2.66$. Figures 3 show the convergence of three algorithms with about 900,000 training steps. The shaded areas are the instantaneous rewards, and the bold lines are the average of the last 100 steps. The algorithms converge around the 400k-th step.

Table I
QoE Metrics When Testing with Both Datasets with $\mu_\phi = 2.66$.
Each Value is the Average Across 10 Seeds. Bold Indicates the Best Result.

| FCC | QoE | Utility | Quality-switch penalty | Rebuffering penalty | Rebuffering time (s) |
|---|---|---|---|---|---|
| A2C | $0.903 \pm 0.104$ | 1.044 | 0.094 | 0.046 | 1.729 |
| DQN | $0.898 \pm 0.047$ | 1.014 | 0.094 | 0.020 | 0.751 |
| PPO | $\mathbf{0.971} \pm 0.046$ | 1.080 | 0.077 | 0.031 | 1.165 |
| THRB | $0.752 \pm 0.439$ | 0.933 | 0.178 | 0.002 | 0.075 |
| BOLA | $0.843 \pm 0.505$ | 0.957 | 0.062 | 0.051 | 1.917 |
| RANDOM | $-2.296 \pm 0.069$ | 0.785 | 0.581 | 2.500 | 50.003 |
| CONSTANT | $-4.224 \pm 0.001$ | 1.160 | 0.020 | 5.364 | 107.283 |
| **LTE** | **QoE** | **Utility** | **Quality-switch penalty** | **Rebuffering penalty** | **Rebuffering time (s)** |
| A2C | $0.600 \pm 0.083$ | 0.966 | 0.106 | 0.260 | 9.774 |
| DQN | $\mathbf{0.616} \pm 0.046$ | 0.915 | 0.145 | 0.152 | 5.741 |
| PPO | $0.590 \pm 0.044$ | 0.978 | 0.136 | 0.251 | 9.436 |
| THRB | $0.537 \pm 0.672$ | 0.841 | 0.194 | 0.267 | 10.03 |
| BOLA | $0.513 \pm 0.942$ | 1.029 | 0.145 | 0.371 | 13.94 |
| RANDOM | $-1.304 \pm 0.023$ | 0.786 | 0.582 | 1.508 | 56.700 |
| CONSTANT | $-1.713 \pm 0.001$ | 1.160 | 0.020 | 2.854 | 107.283 |

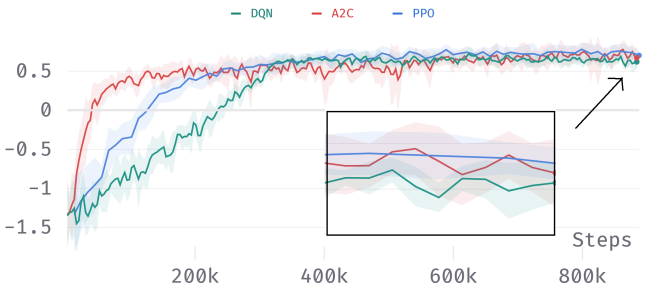



Figure 3. Training curve with $\mu_\phi$ = 2.66. Y-axis is reward value.

Table II
Applying $t$-test to Compare the Mean of Rewards. Bold Indicates $p$-value Less Than 0.05.

| | FCC($\mu_\phi = 2.66$) | | LTE($\mu_\phi = 2.66$) | |
|---|---|---|---|---|
| Alternatives | $t$-value | $p$-value | $t$-value | $p$-value |
| A2C $\neq$ DQN | 0.131 | 0.898 | -0.539 | 0.598 |
| PPO $\neq$ A2C | 1.883 | 0.083 | -0.330 | 0.746 |
| PPO $\neq$ DQN | 3.464 | **0.003** | -1.292 | 0.213 |

From Table I, PPO outperforms A2C and DQN with FCC dataset. However, there is no significant difference between the algorithms with the LTE dataset.

We further investigate the difference in results between the three algorithms by using $t$-test according to the evaluation guidelines suggested by [18] and [17]. We test for the equality of the mean rewards when variances are unknown. Each random sample has 10 data points corresponding to 10 runs with 10 random seeds. The null hypothesis is the equality of the mean rewards of two compared algorithms, and the alternative hypothesis is that the mean rewards are different.

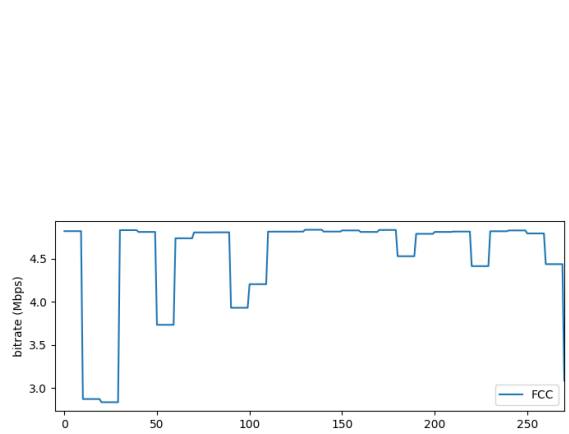
Table II presents $t$-value and $p$-value of various tests between pairs of algorithms. Except for the pair PPO and DQN showing the difference in mean rewards ($p$-value is less than the critical value 0.05), all other $p$-values are greater than 0.05, which means that we can not reject the null hypothesis. Therefore, we can conclude that *there is no significant difference in mean rewards between the RL algorithms*.

Figure 4 shows the quality level of PPO, BOLA, and Throughput of an episode with a randomly picked FCC trace (Figure 4(a)). We can see that PPO yields the best quality. On the other hand, BOLA shows a stable but low quality level. Throughput-based method has a stable buffer level, but the quality levels of the chunks vary.

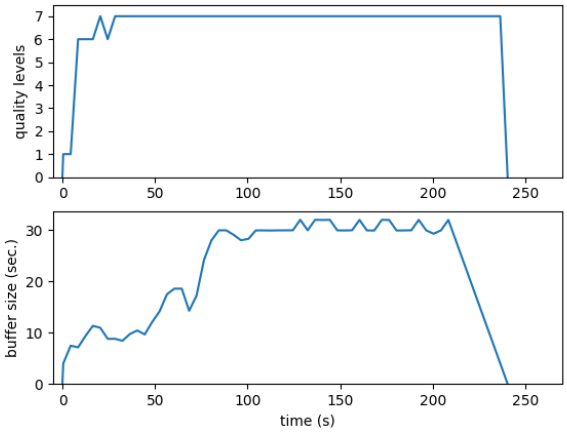
With the DQN, A2C, and PPO models trained for $\mu_\phi = 2.66$, we further test the case $\mu_\phi = 5$ to penalty more on rebuffering time and also to evaluate the robustness of the algorithms. The results are shown in Table III. PPO shows a significantly higher QoE metric in this case, while the rebuffering time is maintained the lowest and smaller than the case $\mu_\phi = 2.66$. This result also agrees with the observation earlier that PPO model is robust against the variations of the hyperparameters; hence, when changing $\mu_\phi$ value without retuning, the performance of PPO model does not degrade as much as A2C and DQN models.

### 4.2 Runtime Analysis

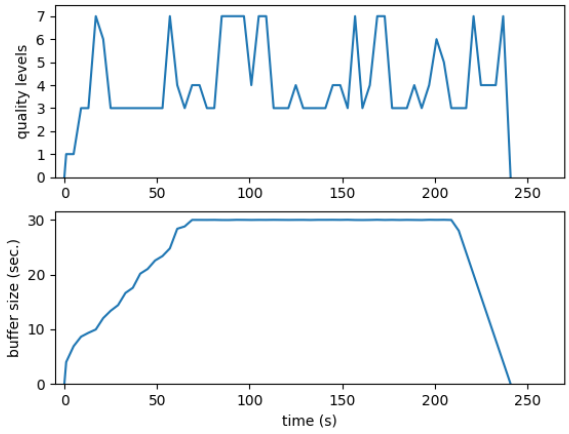
PPO and A2C use two neural networks, whereas DQN uses only the Q-network. Their update rules are also different from each other. Thus, it is important to compare the algorithms' runtime to determine the practical value. From the empirical experiments, each algorithm takes approximately 2 hours on 4 Intel XEON CPU @2.20 GHz and without GPU, and the runtime difference between CPU and GPU is not significant. This indicates that the algorithm is practical to use in real-world situations.



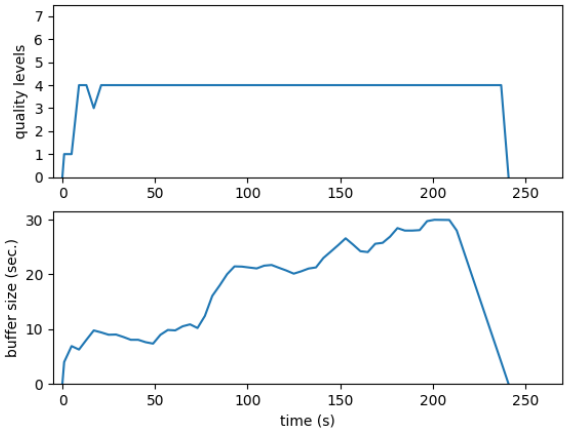

(a) Bandwidth of a randomly chosen trace




(b) PPO




(c) THRB




(d) BOLA

Figure 4. Playing quality levels and buffer sizes of one episode with a randomly picked network trace from the FCC dataset.

Table III
QoE Metrics When Test with Both Datasets in Case $\mu_\phi = 5$.
Each Value is the Average Across 10 Seeds. Bold Indicates the Best Result.

| FCC | QoE | Utility | Quality-switch penalty | Rebuffering penalty | Rebuffering time (s) |
|---|---|---|---|---|---|
| A2C | $0.576 \pm 0.383$ | 0.819 | 0.071 | 0.172 | 3.44 |
| DQN | $0.854 \pm 0.068$ | 1.018 | 0.118 | 0.044 | 0.88 |
| PPO | $\mathbf{0.957} \pm 0.053$ | 1.057 | 0.069 | 0.031 | 0.62 |
| THRB | $0.750 \pm 0.445$ | 0.933 | 0.178 | 0.005 | 0.1 |
| BOLA | $0.798 \pm 0.626$ | 0.957 | 0.062 | 0.096 | 1.92 |
| RANDOM | $-2.296 \pm 0.030$ | 0.786 | 0.582 | 2.500 | 50.003 |
| CONSTANT | $-4.091 \pm 0.002$ | 1.160 | 0.019 | 5.232 | 104.635 |
| **LTE** | **QoE** | **Utility** | **Quality-switch penalty** | **Rebuffering penalty** | **Rebuffering time (s)** |
| A2C | $0.242 \pm 0.304$ | 0.745 | 0.085 | 0.418 | 8.36 |
| DQN | $0.419 \pm 0.095$ | 0.928 | 0.162 | 0.346 | 6.92 |
| PPO | $\mathbf{0.462} \pm 0.048$ | 0.948 | 0.127 | 0.357 | 7.14 |
| THRB | $0.444 \pm 1.064$ | 0.841 | 0.194 | 0.202 | 4.04 |
| BOLA | $0.189 \pm 1.525$ | 1.029 | 0.145 | 0.694 | 13.88 |
| RANDOM | $-2.657 \pm 0.049$ | 0.786 | 0.582 | 2.835 | 56.700 |
| CONSTANT | $-4.224 \pm 0.002$ | 1.160 | 0.020 | 5.364 | 107.283 |

## 5 Conclusions

We have applied three DRL algorithms to bitrate adaptation in DASH, *i.e.*, DQN, A2C, and PPO. We build a simulated environment to train and test with real-trace datasets. The experiments show that the DRL algorithms outperform other rule-based ABR algorithms like throughput-based and BOLA. PPO is more robust against the varying of hyperparameters than the other two DRL algorithms.

## Acknowledgment

This research is funded by Vietnam National University Ho Chi Minh City (VNU-HCM) under grant number DS2020-28-01.

## Appendix

In this section, we describe the hyperparameter tuning strategy. The details of DQN, A2C, and PPO algorithms and the descriptions of their hyperparameters can be found in the works [7, 8, 10].

We use Weights & Biases [26] for experiment tracking and visualizations. The Bayesian tuning strategy is applied to search for the optimal hyperparameters with coefficients $\mu_s = 2.66$ and $\mu_\phi = 2.66$. We perform 120 trials with different seeds for each algorithm. The hyperparameter set from the best trial is then used to train the final model with ten different seeds. Hyperparameters are sampled uniformly in ranges listed in Tables IV, V, and VI.

Table IV
Tuning Ranges of Hyperparameters for DQN.

| **Hyperparameter** | **Data type** | **Sampling distribution** |
|---|---|---|
| Learning rate | float | U(1e-5,1e-3) |
| Experience replay | int | U(59, 11800) |
| Batch size | int | U(59, 590) |
| Learning starts | int | U(295, 2360) |
| Discount factor | float | {0.95, 0.99} |
| Polyak coef | float | {0.99, 1.0} |
| Train frequency | int | U(30, 120) |
| Gradient steps | int | U(-1, 59) |
| Target update interval | int | U(30, 200) |
| Exploration fraction | float | U(0.2, 0.6) |

Table V
Tuning Ranges of Hyperparameters for A2C.

| **Hyperparameter** | **Data type** | **Sampling distribution** |
|---|---|---|
| Learning rate | float | U(1e-5,1e-3) |
| N steps coef | int | U(5, 590) |
| Epoch | int | {10, 20, 30} |
| Discount factor | float | {0.99, 1.0} |
| GAE coef | float | {0.95, 0.99, 1.0} |
| Entropy coef | float | {0, 1e-5, 1e-8} |
| Value function coef | float | U(0.2, 0.5) |
| Use RMSprop | bool | true, false |
| Normalize advantages | bool | true, false |

In addition to the algorithm-specific hyperparameters, each algorithm also has the network hyperparameters presented in Table VII.

Table VI
Tuning Ranges of Hyperparameters for PPO.

| **Hyperparameter** | **Data type** | **Sampling distribution** |
|---|---|---|
| Learning rate | float | U(1e-5,1e-3) |
| Batch size | int | U(59, 590) |
| N steps coef | int | U(1, 5) |
| Epoch | int | {10, 20, 30} |
| Discount factor | float | {0.99, 1.0} |
| GAE coef | float | {0.95, 0.99} |
| Clip range | float | {0.2, 0.3} |
| Entropy coef | float | {0, 1e-5, 1e-8} |
| Value function coef | float | U(0.2, 0.5) |

Table VII
Tuning Ranges of Parameters for Neural Networks.

| **Hyperparameter** | **Type** | **Sampling distribution** |
|---|---|---|
| Features dim | int | {128, 256, 512} |
| Policy network units | int | {64, 128, 256, 512} |
| Policy network layers | int | U(1, 4) |
| Value network units | int | {64, 128, 256, 512} |
| Value network layers | int | U(1, 4) |
| Activation function | str | {tanh, relu} |

Table VIII
Best Hyperparamters of DQN

| Hyperparameter | Value |
|---|---|
| Learning rate | 0.0004295652501295359 |
| Experience replay buffer size | 11340 |
| Learning starts | 1655 |
| Batch size | 181 |
| Polyak update coef | 0.99 |
| Discount factor | 0.99 |
| Model update step (train_freq) | 96 |
| Gradient step | 12 |
| Target Network update interval | 102 |
| Exploration Fraction | 0.36757746500495925 |
| Initial exploration rate | 1.0 |
| Final exploration rate | 0.05 |
| Gradient clipping value | 10 |
| Optimizer | Adam |
| Features dim | 128 |
| Activation function | ReLU |
| Q-network units | 64 |
| Q-network layers | 1 |

The optimal hyperparameters after tuning of DQN, A2C, and PPO algorithms are shown Tables VIII, IX, and X.

Table IX
Best Hyperparameters of A2C

| Hyperparameter | Value |
|---|---|
| Learning rate | 0.00011583245926619557 |
| Experiences per update | 7 |
| Discount factor | 1 |
| GAE lambda | 0.95 |
| Entropy coef | 1e-8 |
| Value function coef | 0.32803323811081575 |
| Gradient clipping value | 0.5 |
| Use RMSProp | True |
| Normalize advantage | True |
| Features dim | 256 |
| Activation function | Tanh |
| Policy network units | 64 |
| Policy network layers | 1 |
| Value network units | 128 |
| Value network layers | 1 |

Table X
Best Hyperparameters of PPO

| Hyperparameter | Value |
|---|---|
| Learning rate | 0.00012259927557284572 |
| Experiences per update | 2875 |
| Batch size | 556 |
| N_steps coef | 5 |
| Update epoch | 10 |
| Discount factor | 1 |
| GAE lambda | 0.95 |
| Clip range | 0.3 |
| Entropy coef | 1e-8 |
| Value function coef | 0.4348768251443197 |
| Gradient clipping value | 0.5 |
| Optimizer | Adam |
| Features dim | 512 |
| Activation function | ReLU |
| Policy network units | 512 |
| Policy network layers | 4 |
| Value network units | 128 |
| Value network layers | 3 |

## References


[1] Cisco, "Cisco visual networking index: Forecast and methodology 2015–2020," 2016.

[2] T. Stockhammer, "Dynamic adaptive streaming over http– standards and design principles," in *Proceedings of the second annual ACM Conference on Multimedia Systems*, 2011, pp. 133–144.

[3] T.-Y. Huang, N. Handigol, B. Heller, N. McKeown, and R. Johari, "Confused, timid, and unstable: picking a video streaming rate is hard," in *Proceedings of the 2012 Internet Measurement Conference*, 2012, pp. 225–238.

[4] X. K. Zou, J. Erman, V. Gopalakrishnan, E. Halepovic, R. Jana, X. Jin, J. Rexford, and R. K. Sinha, "Can accurate predictions improve video streaming in cellular networks?" in *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications*, 2015, pp. 57–62.

[5] K. Spiteri, R. Urgaonkar, and R. K. Sitaraman, "BOLA: Near-optimal bitrate adaptation for online videos," *IEEE/ACM Transactions on Networking*, vol. 28, no. 4, pp. 1698–1711, 2020.

[6] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli, "A control-theoretic approach for dynamic adaptive video streaming over http," in *Proceedings of the ACM Conference on Special Interest Group on Data Communication*, 2015, pp. 325–338.

[7] 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.

[8] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *Proceedings of the 33rd International Conference on Machine Learning*. PMLR, 2016, pp. 1928–1937.

[9] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *Proceedings of the International Conference on Machine Learning*. PMLR, 2015, pp. 1889–1897.

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

[11] H. Mao, R. Netravali, and M. Alizadeh, "Neural adaptive video streaming with pensieve," in *Proceedings of the conference of the ACM Special Interest Group on Data Communication*, 2017, pp. 197–210.

[12] M. Gadaleta, F. Chiariotti, M. Rossi, and A. Zanella, "D-DASH: A deep Q-learning framework for DASH video streaming," *IEEE Transactions on Cognitive Communications and Networking*, vol. 3, no. 4, pp. 703–718, 2017.

[13] Blender Elephants Dream Movie, 2014. [Online]. Available: https://orange.blender.org/

[14] Google, "Choose live encoder settings, bitrates, and resolutions," *YouTube help*, 2021. [Online]. Available: https://support.google.com/youtube/answer/2853702

[15] D. Raca, J. J. Quinlan, A. H. Zahran, and C. J. Sreenan, "Beyond throughput: A 4G LTE dataset with channel and context metrics," in *Proceedings of the 9th ACM Multimedia Systems Conference*, 2018, pp. 460–465.

[16] FCC, "The Tenth Measuring Broadband America Fixed Broadband Report: A Report on Consumer Fixed Broadband Performance in the United States," 2019.

[17] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018.

[18] C. Colas, O. Sigaud, and P.-Y. Oudeyer, "A hitchhiker's guide to statistical comparisons of reinforcement learning algorithms," *arXiv preprint arXiv:1904.06979*, 2019.

[19] L. Engstrom, A. Ilyas, S. Santurkar, D. Tsipras, F. Janoos, L. Rudolph, and A. Madry, "Implementation matters in Deep RL: A case study on PPO and TRPO," in *Proceedings of the International Conference on Learning Representations*, 2019.

[20] R. Islam, P. Henderson, M. Gomrokchi, and D. Precup, "Reproducibility of benchmarked deep reinforcement learning tasks for continuous control," *arXiv preprint arXiv:1708.04133*, 2017.

[21] A. Raffin, A. Hill, M. Ernestus, A. Gleave, A. Kanervisto, and N. Dormann, "Stable baselines3," *GitHub Repository*, 2019. [Online]. Available: https://github.com/DLR-RM/stable-baselines3

[22] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[23] G. Hinton and T. Tieleman, *Neural networks for machine learning class*. Coursera, 2012.

[24] https://wandb.ai/aeryss/singlepath final.

[25] https://wandb.ai/aeryss/singlepath tuning.

[26] L. Biewald, "Experiment tracking with weights and biases," 2020. [Online]. Available: https://www.wandb.com/

**Long Minh Luu** holds a bachelor's degree of Computer Science from International University - Vietnam National University Ho Chi Minh City. His main research areas are Reinforcement Learning and Computer Vision for bitrate adaptation, image classification, and out-of-distribution generalization.

**Nghia Trung Nguyen** received his bachelor's and master's degreesfrom Computer Science from International University - Vietnam National University Ho Chi Minh City in 2019 and 2022, respectively. His main research interest is to apply Reinforcement Learning to enhance the efficiency of various applications in computing.

**Phuong Luu Vo** received her B.Eng and M.Eng degrees in electrical-electronics engineering from Ho Chi Minh City University of Technology, Vietnam in 1998, 2002, respectively, and Ph.D. degree at Kyung Hee University, Korea in 2014. Currently, she is an Associate Professor at School of Computer Science and Engineering at International University – VNUHCM. Her research interest is to apply machine learning, optimization, and game theory to contemporary networks.

**Tuan-Anh Le** received Ph.D. degree at Kyung Hee University, Korea in 2013. Currently, he is with Faculty of Engineering and Technology, Thu Dau Mot University, Vietnam. His research interest is to apply artificial intelligent to contemporary networks.