Applied Reinforcement-Learning in Adaptive Bitrate Streaming

Garrett Hurst & Ron Bejerano

Purdue University School of Electrical and Computer Engineering
ECE59595: Rienforcement Learning
gahurst@purdue.edu, rbejeran@purdue.edu

Abstract

This study explores the application of reinforcement learning (RL) in enhancing Adaptive Bitrate (ABR) video streaming over HTTP. Utilizing real-world network traces and open-source simulation tools, RL models outperform traditional ABR approaches, achieving significant reductions in rebuffering and bitrate variability. Despite the advancements, challenges in balancing QoE components remain. This work underscores RL's promise in ABR optimization and sets the stage for future developments in robust video streaming solutions.

Code —
https://github.com/g-hurst/RL-based-ABR-streaming

Objective

The objective of this project is to apply reinforcement learning in video delivery over HTTP. Video streaming services use Adaptive Bitrate (ABR) streaming to allow the quality of a video stream to be dynamically adjusted based on the current network conditions, device capabilities, and other factors. This study aims to improve existing ABR algorithms by applying reinforcement learning to the quality selection process.

Motivation

Whether it be from movies and TV shows to social media, video streaming is ubiquitous in today's modern world. Furthermore, video quality has been shown to directly affect user engagement with content. Thus, optimizing the quality of experience of video streaming for a user ultimately improves the revenue of streaming providers (Dobrian et al. 2011). With estimates projecting video streaming to explode in demand, potential research into the confluence of ABR and RL stands to further improve the medium of video streaming in terms of efficiency and quality of experience.

Introduction & Background

Due to the highly complex and changing structure of the Internet, streaming video over HTTP is plagued with network instability. Adaptive Bitrate (ABR) streaming is a method used to enhance playback quality by automatically adjusting the video quality in response to constantly changing real-time network conditions. ABR achieves this by encoding video content at multiple quality levels, each with a different bitrate. As the user streams the video, the ABR algorithm monitors their network's speed and stability, seamlessly switching between quality levels in order to optimize a quality of experience (QoE) factor. For example, if a user's network becomes congested, ABR will switch to a lower quality to avoid buffering; if the connection improves, it can return to a higher resolution.

On the server side, supporting ABR algorithms requires several processes to prepare, store, and deliver video content efficiently. Video content is segmented into smaller segments that typically range from 2 to 10 seconds. Then each chunk is encoded at multiple quality levels that correspond to different bitrates. Furthermore, the size of a stored chunk also depends on the content itself. For example, a two-second 4K clip with content that captures a moving action scene would require more storage space than a two second 4K clip of a still landscape.

On the user side, it is important to note that there is no standard metric for user QoE. However, while the exact definition of QoE is subject to the desires of streaming providers, it has been shown that several factors such as average video quality (viewing at a higher resolution), number of quality swaps, and number of buffering occurrences directly affect the user experience (Dobrian et al. 2011). Due to this, these three factors were chosen to define QoE in this paper.

Literature Review

ABR Algorithms

Research on ABR algorithms has been a prevalent topic within the communications community for over a decade now. Previous approaches have made efforts to optimize throughput estimations and chunk download scheduling, while others have taken rate-based or buffer-based approaches. Many of which model the task as an optimal control challenge (Yin et al. 2015; Spiteri, Urgaonkar, and Sitaraman 2020).

MPC: Model Predictive Control

Traditional approaches to ABR algorithms have either taken rate-based or buffer-based approaches to model optimal bitrate selection. The MPC algorithm takes advantage of both both rate information and buffer information in order to model an optimal bitrate section over a look ahead window (Yin et al. 2015). While this approach is now dated, it heavily inspired many of the current ABR algorithms such as BOLA, BOAL-E, and DYNAMIC which are industry-standard in many common applications.

BOLA: Buffer Occupancy Lyapunov Algorithm

One of the most common open-source video players is the DASH.js client. The current DASH client uses an ABR algorithm called BOLA that was developed in a partnership between the University of Massachusetts at Amherst and Amazon Prime Video. BOLA stands for the Buffer Occupancy Lyapunov Algorithm. The primary objective of the algorithm is to maximize a linear combination of two metrics: the average playback quality in terms of bitrate and the percentage of time spent not buffering. To achieve this objective, BOLA takes a buffer-centric approach by relying on the buffer occupancy metric and then tailoring the algorithm to prioritize certain conditions, such as higher video quality or minimizing re-buffering events. This approach separates BOLA from other ABR algorithms as it does not depend heavily on throughput estimation and prediction and rather focuses on actual data at its core. (Spiteri, Urgaonkar, and Sitaraman 2020).

Through experimentation, BOLA demonstrated some key advantages over other leading ABR algorithms. The first is its improved playback stability. The algorithm minimized bitrate oscillations and prioritized steady quality levels, which ultimately resulted in a smoother viewing experience regardless of the network conditions. Secondly, the simplicity of the model to rely solely on the buffer occupancy and disregard more computationally expensive calculations and predictions of throughput enabled easy integration within existing systems and was particularly advantageous for devices with limited processing power. However, while this approach yields good performance, it solely relies on observations of the video buffer which leave out potentially valuable rate information. Previous ABR algorithms such as MPC have shown that observation of both rate and buffer can lead to improvements on user QoE (Yin et al. 2015). In addition, the model treats all bitrate levels equally spaced in regard to quality improvement which may differ against the perceived quality metrics. It may also struggle with extreme network scenarios such as high latency or frequent packet loss. Despite these shortcomings, BOLA has become the industry standard and has been implemented within the DASH.js client. Its simplicity and easily integratable system within edge computing environments offer a compelling alternative to throughput-predictive and heuristic-based ABR strategies while prioritizing an optimal viewing experience.

DYNAMIC Algorithm

One of the main points of contention with the BOLA algorithm is its rigidity in its core structure. The reasoning for why the algorithm ony considers buffer occupancy metrics as the basis for its bitrate selection method is sound and ensures a smoother video quality, but it does leave the model susceptible to varying and frequent network conditional changes. Should the bandwidth unexpectedly spike or drop, BOLA will be unable to take advantage or deploy countermeasures effectively to keep video quality stable. One such solution to this issue is an extension of BOLA simply termed DYNAMIC.

DYNAMIC is a hybrid ABR algorithm that combines BOLA and traditional throughput-predictive models based on the video buffer level. If the buffer level is low, the typical network conditions suggest that the system is on startup or seek, both of which perform better under throughput-predictive models. However, when the buffer level is high, it suggests that the network has provided consistent bandwidth and low latency conditions, which better suit the stability of BOLA. As a result, the DYNAMIC algorithm institutes a switching rule that should predictive throughput exceed 10 seconds or more, the algorithm will switch to a BOLA-based model, whereas if the conditions are opposite, the heuristic-based throughput predictive algorithms will kick in (Spiteri, Sitaraman, and Sparacio 2018). Combining these two provides the DYNAMIC algorithm with the best of both worlds in terms of ABR models and results in improved efficiency and video streaming quality across a wider range of network conditions, both favorable and unfavorable.

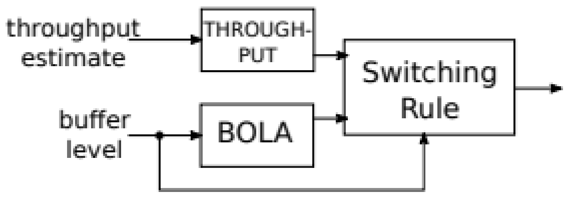

Figure 1: DYNAMIC Algorithm Diagram (Spiteri, Sitaraman, and Sparacio 2018)

With this knowledge, recent designs of the RL-based ABR algorithm have shown potential to outperform traditional controls-based approaches (Wei et al. 2021). However, the definitions of QoE are more convoluted than historical approaches, and the Markov Decision Processes used to define the streaming problem often contain limited information about known future states. RL-based approaches are increasingly exploring hybrid methods that combine buffer-based strategies with predictive modeling to address these challenges. These developments aim to refine QoE metrics further while leveraging reinforcement learning's adaptability to dynamic and uncertain network environments.

Environments and Simulators

In addition to the development of the BOLA and DYNAMIC ABR algorithms, the developers also released an open source ABR environment simulator called Sabre.

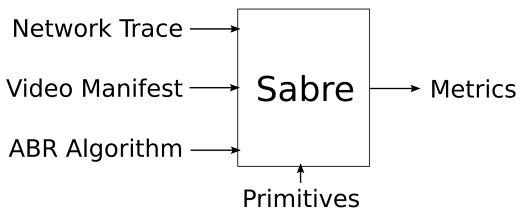

Figure 2: Sabre Inputs & Outputs (Spiteri, Sitaraman, and Sparacio 2018)

Sabre emulates an ABR environment based on several inputs shown in Figure 2. A network trace allows allows for simulation of the network conditions that the ABR algorithm operates under. This is given in json format and contains the network bandwidth and latency over time. A video manifest contains the all the encoding information that a video to be streamed would contain. This includes the duration of segments, bitrate options, and size of each chunk at the given bitrates. The manifest format used in Sabre parallels the standard manifest format used in the DASH player. Finally, the ABR algorithm is input as a python file that contains a class with the same name as the file. The class inherits from the Sabre ABR class and overloads basic methods in order to define how the algorithm selects each bitrate under the simulated conditions (Spiteri, Sitaraman, and Sparacio 2018).

Methodology

Model

The interaction between an agent choosing bitrates and a user streaming environment can be modeled as a Markov Decision Process (MDP) consisting of five main elements:

$$\text{MDP} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma \rangle$$

- State Space ($\mathcal{S}$): Streaming Environment Characteristics
- Action Space ($\mathcal{A}$): Bitrate Selection
- Transition Probability Function ($\mathcal{P}$): Next State Probabilities Given Current Network Conditions
- Reward Function (r): User QoE
- Discount Factor ($\gamma \in (0, 1]$): Prioritizes Immediate vs. Future Rewards

State Space:

The state space was chosen to capture several factors that affect QoE. The most immediate factor is the selection of the bitrate to download the next immediate chunk. This bitrate is used to select which size option in the video manifest to download. Another factor represented that affects QoE is the current state of the downloaded buffer; more video contained within the buffer leaves more time to download larger segments in the future. Finally, network information such as the current throughput as well as mean and variance throughput are valuable in estimating the time required to download future segments. The last factor in the state space is a look ahead of the future segment download size opitons. This gives the ability to plan for future chunks that may be smaller or larger.

In better mathematical notation, the state space can be represented as a tuple containing the following elements:

$$S_n = (q_n, \{q_{n+1}, ..., q_{n+4}\}, B_n, C_n, L_n)$$

- q_n are the n'th chunk size options in Bytes that are associated with each bitrate
- $\{q_{n+1}, ..., q_{n+4}\}$ are look ahead chunk sizes in Bytes
- B_n is the current buffer capacity in seconds
- C_n is a 3-tuple to represent the current throughput state. It is composed of: the currently observed network throughput in Bytes per second, the mean of the past 5 throughput observations, and the standard deviation of past 5 throughput observations
- L_n is a 3-tule to represent the current latency state. It is composed of: the currently observed network latency in seconds, the mean of the past 5 latency observations, and the standard deviation of past 5 latency observations

Action Space:

The action space is defined as the bitrates that a the user can select to download at the current chunk. This can be represented as a discrete set of size N_q, where N_q is the number of bitrate options that the user may select from.

Transition Probability Function:

In this experiment, the transition probability of going from one state to another is dependent only on the action that is taken. Furthermore, when given a bitrate selection action, the probability of tranisitioning to that bitrate in the next state is 1.

Reward Function:

Because QoE is the value that we would like to optimize, it can be directly used as the reward function. More formally the average user QoE over a video is represented as:

$$\text{QoE}_i = \alpha B_i - \beta |B_i - B_{i-1}| - \delta t_i$$

- B_i is the selected bitrate at step i
- t_i is the amount of time that the video buffered at step i
- $\alpha, \beta, \text{and}, \delta$ are weights given to each metric based on the important to the streaming provider.

Defining QoE in this way rewards is intended to maximize the selected bitrate while minimizing the change in bitrate and time spent rebuffering. Thus, the given reward function can be defined as:

$$r_i(s_i, a_i) = QoE_i \qquad \text{for a single chunk } i$$

$$R_i = \sum_{n=0}^{N_v - i} \gamma^n r_{i+n}(s_{i+n}, a_{i+n}) \quad \text{for the remaining chunks}$$

In this case, i represents the i'th chunk of a video to be downloaded, and N_v represents the total number of chunks in the video.

Candidate RL Solutions

Advantage Actor Critic (A2C):

A2C is a model-free on-policy algorithm that synchronously updates a centralized policy through the use of an advantage function. This model extends elements taken from the standard actor-critic (AC) algorithms and applies the advantage function to focus on increasing return actions to strike a balance between policy exploration and exploitation.

In a typical actor-critic algorithm, two neural networks are developed in tandem: the actor, and the critic, and both have their separate roles in reaching the optimal policy. The actor acts as the main agent of the model. It selects actions based on the current policy and aims to achieve maximum cumulative rewards while adapting to any environmental changes. In this way, the actor is trained on a policy network similar to an agent trained through the REINFORCE algorithm. The critic evaluates the performance of the actor by estimating the value or quality of the actions chosen by the actor and providing feedback to the actor to then guide the network towards higher expected returns. The critic is trained on a state-value network in a similar fashion to an agent as part of a Deep Q-Learning (DQN) network. Together, the actor and critic combine to form a model that learns incrementally, requires no replay buffer through its synchronous nature of multiple workers, and produces a stable training return.

The actor-critic algorithm has inspired many different offshoots and models that branch from its simplistic structure. One of the most promising is Advantage Actor-Critic (A2C) which combines the standard actor-critic formula with an advantage function. The advantage function provides a measure of how much better or worse an action is compared to the value return of the average action through the entire action space. By providing another metric to which the neural networks can compare their results with, the model can focus on actions with returns that have the largest positive difference from actions with the average return rather than updating from the difference between the critic's estimated value and the true return value. A2C works particularly well in discrete action spaces, which simplifies from the complexity of other models which may consider wider action spaces in continuous forms. Unlike its asynchronous alternative (A3C), A2C updates its actor incrementally all at the same time from its multiple workers, which promotes the stability of having a predictable influx of data at certain moments as opposed to random influxes of data from out of sync sources and also provides a more cost-effective and efficient algorithm to run. Combining these benefits produces a model that is flexible yet stable while performing under peak efficiency. (Mnih et al. 2016)

Algorithm 1: A3C Algorithm per Thread (Mnih et al. 2016)

```
Initialize thread step counters $t \leftarrow 0, T = 0$
Initialize target network weights $\theta^- \leftarrow \theta$
Initialize network gradients $d\theta \leftarrow 0$
Get initial state $s$
while $T \leq T_{max}$ do
    Select action $a$ with $\epsilon$-greedy policy from $Q(s, a|\theta)$
    Execute $a$ and observe $r$, $s'$
    $y = \begin{cases} r, \text{ for terminal } s' \\ r + \gamma \max_{a'} Q(s', a'|\theta^-), \text{ for non terminal } s' \end{cases}$    (1)
    Accumulate gradients $d\theta \leftarrow d\theta + \frac{\partial(y - Q(s,a|\theta))^2}{\partial\theta}$
    Set $s = s'$
    $T \leftarrow T + 1, t \leftarrow t + 1$
    if $T \% I_{target} == 0$ then
        Update target network $\theta^- \leftarrow \theta$
    end if
    if $T \% I_{update} == 0$ or $s$ is terminal then
        Perform update of $\theta$ using $d\theta$
        Clear gradients $d\theta \leftarrow 0$
    end if
end while
```

A2C provides a strong candidate as a solution to an ABR streaming algorithm. The model is able to flexibly select between discrete bitrates and balance the quality of the streaming product with the computational complexity and efficiency required with the limited resources available. Typical ABR environments have frequent network changes and additional preferences that it must be able to leverage and utilize a model that is capable of accepting rapid and varying inputs and produce a consistent output. A2C provides that solution and does so in a quick and responsive manner and ensures that the product is as seamless as possible.

Deep Deterministic Policy Gradient (DDPG):

DDPG is a model-free off-policy model that is capable of adapting continuous action spaces by making the Q network differentiable with respect to the target policy. The model combines elements from both Deterministic Policy Gradient (DPG) and Deep Q-Learning (DQN) in that it applies an actor-critic (AC) approach to DPG to concurrently learn the Q function and the policy.

The main benefits of using DDPG is its simplicity and efficiency in arriving towards the optimal policy. Since the model is deterministic, unlike other actor-critic models, every state is mapped to an action. This differs from the stochastic approach of other models, in which an action is determined by the probability distribution of all actions based on the current state. This simplifies the model as it is guaranteed that if a state is reached, that the next action will be known, thus making the model capable of mapping continuous action spaces. The model also benefits from having an off-policy nature. In traditional on-policy models, the policy is iteratively refined to then generate control actions within the environment. This is a behavioral policy. Off-policy models also utilize a behavioral policy, but in conjunction with a target policy, which is being trained to optimize the behavioral policy towards a final convergent system. Off-policy models benefit from having two separate policies being trained simultaneously. For one, off-policy can result in parallel learning, both speeding up the time it takes for the model to converge and also having continuous exploration where one policy is learning the for its optimal point while the other continues to explore potential other actions and states. It also benefits by not requiring the periodic update of all of the weights in its networks. It can update a certain portion of the weights at a time instead of all of the weights like one would see in an on-policy model (Lillicrap et al. 2019).

Algorithm 2: DDPG Algorithm (Lillicrap et al. 2019)

```
Initialize critic $Q(s, a|\theta^Q)$ with weight $\theta^Q$
Initialize actor $\mu(s|\theta^\mu)$ with weight $\theta^\mu$
Initialize target network $Q'$ with weight $\theta^{Q'} \leftarrow \theta^Q$
Initialize target policy $\mu'$ with weight $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer $\mathcal{R}$
**for** $episode = 1$ **do**
    Initialize process $\mathcal{N}$ for action exploration
    Receive initial observation state $s_1$
    **for** $t = 1$ **do**
        Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$
        Execute $a_t$ and observe $r_t$, $s_{t+1}$
        Store $(s_t, a_t, r_t, s_{t+1})$ in $\mathcal{R}$
        Sample batch of $N$ transitions from $\mathcal{R}$
        Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
        Set loss $L = \frac{1}{N}\Sigma_i(y_i - Q(s_i, a_i|\theta^Q))^2$
        $\nabla_{\theta^\mu}\mathcal{J} \approx \frac{1}{N}\Sigma_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i}$
        Set $\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$
        Set $\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$
    **end for**
**end for**
```

In terms of the compatibility of DDPG with an ABR focus, the model lends itself well towards constraining to the state and action spaces that optimize the QoE factor. In typical ABR environments, the algorithm requires rapid decision making in order to preserve the best streaming quality possible considering the network conditions. These network conditions can vary continuously, and when paired with buffer status and the viewer's own preferences at any given moment, any ABR model must be adaptable on a dime while continuing to produce stable output. DDPG accomplishes this goal through its properties of being off-policy and deterministic, as it affords the model flexibility in adapting to new changes in the environment while simplifying the potential next state output to a single guaranteed state when considering a current state and action based on an optimal policy. DDPG places stability at the forefront of its objectives and as such offers the best viewing experience for clients and users through ABR streaming.

Project & Plan Outcomes

Formalized Problem

The RL problem can be formally defined as finding an optimal policy π^* to select bitrates that maximize the user QoE_{avg}. With MDP defined above, the goal of this problem can be represented with a the maximum J value at each state. This can be more formally defined as:

$$\pi^*(s) = \text{argmax}_\pi J(s, a) \quad \forall s \in \mathcal{S}$$

In this case, the objective function, $J(s, a)$, is defined in two ways dependent on the model. With the A2C model, J is the advantage-actor critic function used to estimate the policy. With the DDPG model, J is a differential $Q-$learning function that is directly applied to the policy.

Data

The data chosen for this experiment was the same set that was used to develop the BOLA algorithm (Spiteri, Urgaonkar, and Sitaraman 2020). The video manifest used was from the Big Buck Bunny movie (Blender 2008). To model a variety of network environments, a total of three sets of traces were used. They are defined as follows:

1) A set of 86 3G mobile network traces from from public transportation routes in Norway. Traces have a granularity of 1 second (uMass 2013).
2) A set of 40 4G mobile network traces publicly collected in Belgium. Once again, traces have a granularity of 1 second (van der Hooft et al. 2016).
3) A set of 2000 broadband network traces from the US FCC. Traces have a granularity of 5s. The traces are also divided into two subsets of 1000 HD traces and 1000 SD traces (Commission 2016).

While the full dataset is described above, only a subset of the data was utilized to test and train each model. Training with the full dataset required nearly half a day of computation time which would not make it feasible to tune hyper parameters. In the case of future work, using the full dataset would be a great start, so the full set is intentionally left in the code repository for this experiment.

To train and test the model, the subset of 1000 HD traces from the US FCC set was used. The set was divided randomly into 750 traces for training and 250 traces for testing.

Model Specification

The models were implemented in with the help of several open-source libraries. The Sabre ABR algorithm simulator was used to both train and test the models under simulated ABR streaming conditions. The OpenAI Gymnasium library was used to define the enthronement. Stable-Baselines3 was used to create instances of a DDPG and A2C agent that were trained on interactions with the environment.

Interacting with Sabre:

Two classes were created in order to allow the RL models to interact with the Sabre emulator. First, an Emulator wrapper class was created to take a video manifest and network trace as inputs. This allowed for multiple instances of the class to be created to run multiple simulations with ease. The Emulator sets up Sabre, and then provides a step method that takes a bitrate selection and returns the throughput, latency, and rebuffer time the is observed when downloading the chunk at selected bitrate.

The other class created was a custom environment that extends the OpenAI Gymnasium Environment class. This class takes the movie manifest and a list of all the network traces as an input. In order to generate trajectories, this class also defines the transition function for the MDP. For each video chunk, the environment class takes an action and makes an observation by collecting the state of the Emulator class. Once no more chunks remain in the video, the trajectory reaches a terminal state, and a new emulator is created with the next network trace.

Hyper-parameter Tuning:

Both the A2C and DDPG models were trained with following (α, β, δ) reward function parameters:

$$r_i = \text{QoE}_i = \alpha B_i - \beta|B_i - B_{i-1}| - \delta t_i$$

$$(\alpha, \beta, \delta) = (\frac{1}{N_q}, \frac{1}{4 \cdot N_q}, \frac{N_v}{7 \cdot \text{segment\_length}})$$

Recall from the model methodology, that N_q is the number of bitrate quality options to select from at each chunk (the size of the action space), and N_v represents the total number of chunks in the video. The α parameter was chosen to divide the current bitrate by the total number of bitrate options. This serves to normalize the bitrate selection to $alpha * B_n \in [0, 1]$. The β parameter normalizes the change in bitrate the same as α, but then decreases the range to be $\beta|B_n - B_{n-1}| \in [0, 0.25]$. This reduced range makes it such that β has 25% of the impact on the reward function when compared to α.

The δ parameter was chosen to cause the cumulative reward of playing the movie to be zero whenever the video buffered for more than 7 segments of time. Dividing the buffer time by $(7 * segment\_length)$ scales it to be the percentage of rebuffer time over 7 segments. Then, multiplying this by the number of segments ensures that the reward gained from $\alpha * B_n$ reaches zero if the video buffers for 7 segments of time.

Many of the model-specific hyper parameters were left as the default kwargs from the Stable-Baselines3 class object. The parameters for the A2C model are:

- Discount factor (γ) = 0.99
- Value function loss coefficient = 0.5
- Gradient clip = 0.5
- Optimizer = RMSprop

The parameters for the DDPG model are:

- Discount factor (γ) = 0.99
- Learning rate: 0.001
- Buffer size: 1,000,000
- Batch size: 256
- Gradient steps per rollout: 1

Results

The results collected were generated by testing each algorithm over 250 unseen traces from the 1000 HD traces in the US FCC dataset. In addition to the A2C and DDPG models, results from the Dynamic and Bola algorithms were collected to serve as a baseline to compare against. The CDF plots in Figure 3-5 show the total time spent rebuffering, total bitrate change, and total played bitrate respectively; the same information is also presented as a histogram in Figures 6-8.

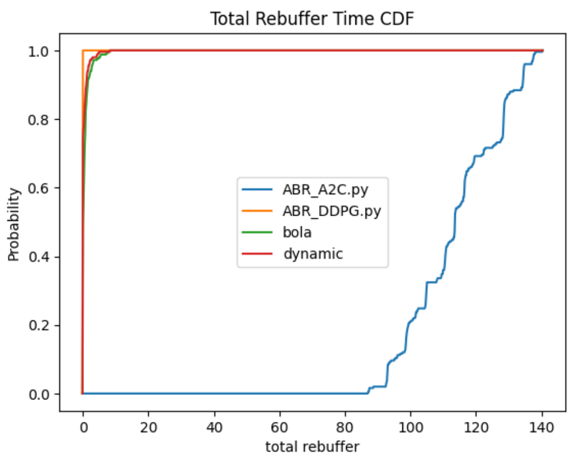

Figure 3: Rebuffer Time Cumulative Distribution

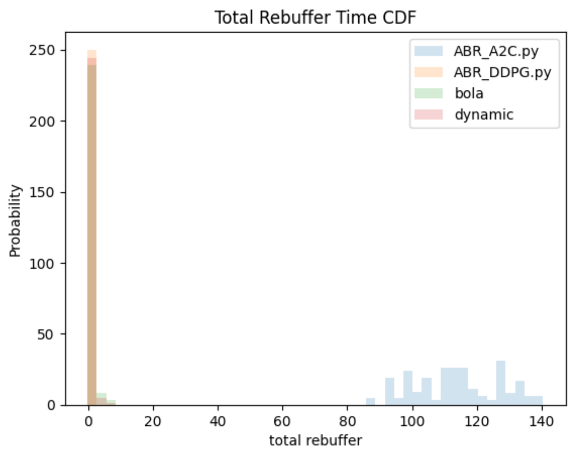

Figure 6: Rebuffer Time Histogram

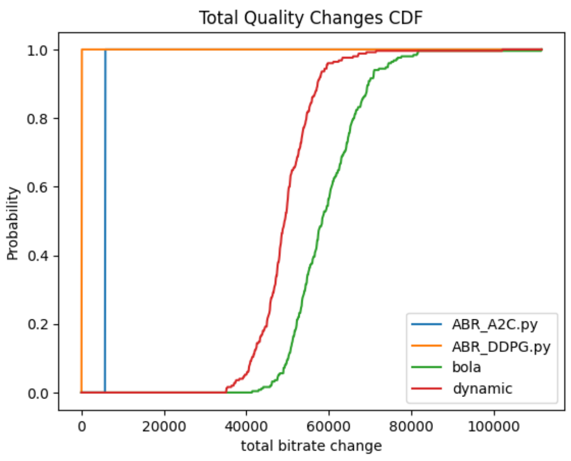

Figure 4: Quality Change Cumulative Distribution

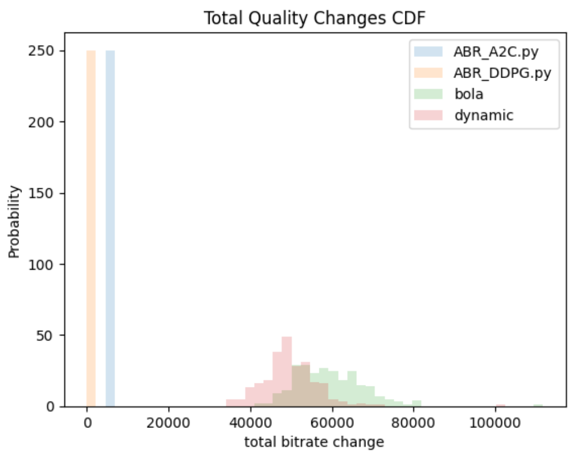

Figure 7: Quality Change Histogram

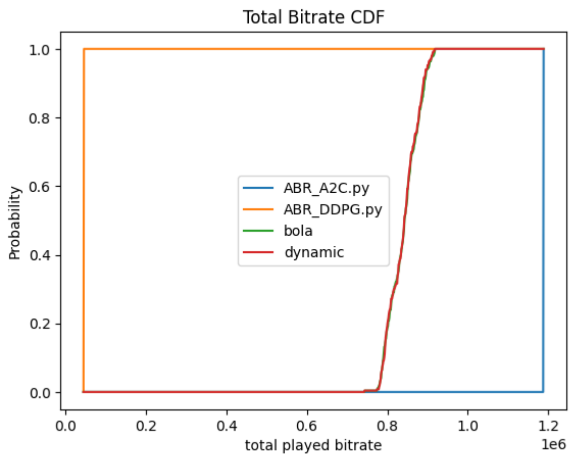

Figure 5: Total Bitrate Cumulative Distribution

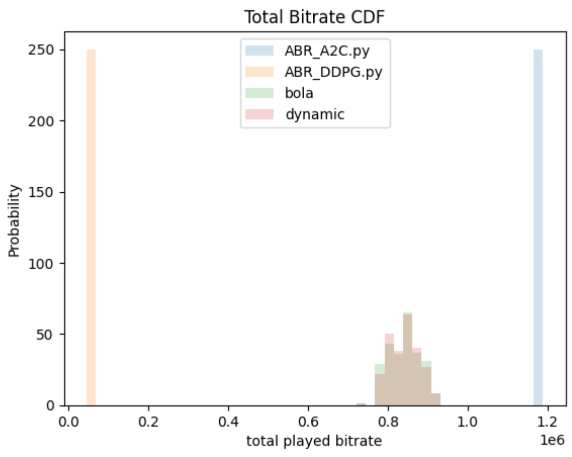

Figure 8: Total Bitrate Histogram

The time each algorithm spent rebuffering is presented in Figure 3 and Figure 6. The A2C model spent the most time rebuffering, and it also had the highest variance. The DDPG model spent nearly zero time rebuffering across all traces and rebuffered for less time than both of the base algorithms, Bola and Dynamic.

The total bitrate change is shown in Figure 4 and Figure 7. This measurement was a summation of the bitrate difference between each successive movie chunk. It can be seen that both RL models had minimal changes in quality and a low variance of quality changes across traces when compared to the two base algorithms.

Finally, the total bitrate is shown in Figure 5 and Figure 8. The total bitrate was calculated by summing the chosen bitrate for every movie chunk. The base algorithms performed near identically, A2C selected the highest bitrate across all chunks, and DDPG selected the lowest bitrate across all chunks.

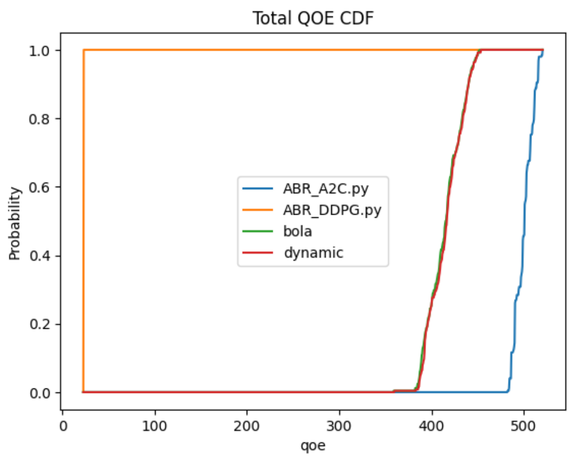

Figure 9: Quality of Experience Cumulative Distribution

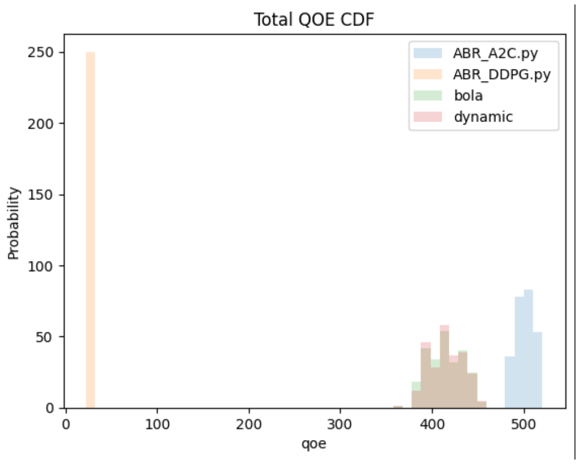

Figure 10: Quality of Experience Histogram

The total QoE was then plotted with the information from Figures 3-8 to generate Figure 9 and Figure 10. The QoE for the DDPG model was by far the lowest at nearly zero. The two base algorithms performed as expected with similar distributions. The A2C model delivered the highest user QoE.

Algorithm	A2C	DDPG	bola	dynamic
Rebuffer Time	113	**0**	0.57	0.32
Bitrate Changes	5,770	**0**	59,469	49,719
Bitrate Total	118,8230	45,770	**841,882**	841,083
QOE	**500**	22	414	415

Table 1: Model Performance

Finally, a simple mean was taken from each metric and is listed in Table 1.The model that performed is listed in bold.

Discussion

The results provide significant insights into the performance of the A2C, DDPG, Bola, and Dynamic algorithms based on rebuffer time, total bitrate, quality changes, and overall quality of experience (QoE). The evaluation spans over 250 unseen traces, showcasing varying performance metrics:

Rebuffer Time:

The cumulative distribution (CDF) and histogram clearly illustrate the differences in rebuffering behavior between the algorithms. The A2C model shows the highest average rebuffering time (113 seconds) with considerable variance, suggesting its aggressive prioritization of higher bitrates. In contrast, the DDPG model achieves near-zero rebuffering time, outperforming even the base algorithms (Bola and Dynamic). This indicates that DDPG focuses more on stabilizing playback rather than achieving higher QoE or bitrates. Since changing bitrates reduces the reward, it is possible that this reward reduction prevented the DDPG model from exploring the options of selecting higher bitrates.

Quality Changes:

Both RL-based models performed well in minimizing quality changes compared to Bola and Dynamic, as shown in Figures 4 and 7. This reduction in quality shifts likely contributes to improved user-perceived stability. Notably, DDPG records the lowest total quality changes, further emphasizing its focus on a consistent streaming experience. However, one thing to note is that this consistency in bitrate selection was at the expense of selecting higher bitrates. In contrast, the Bola and Dynamic algorithms exhibit significantly higher quality variability, which reduces from user experience.

Total Bitrate:

A2C consistently selects the highest total bitrate, as evidenced by the CDF and histogram plots. This aligns with its higher QoE scores but comes at the expense of increased rebuffering. DDPG, on the other hand, selects the lowest bitrate, prioritizing smooth playback over video quality. Bola and Dynamic display nearly identical bitrate distributions.

Quality of Experience:

The QoE metric consolidates the individual factors into a holistic measure of user satisfaction. A2C achieves the highest QoE, averaging 500, reflecting its prioritization of high-quality playback. DDPG's QoE is significantly lower, at just 22, despite its minimal rebuffering and stable quality, indicating that its conservative bitrate selection compromises user satisfaction. Bola and Dynamic achieve moderate QoE values (414-415), balancing quality and stability.

Conclusion & Future Work

The results highlight the trade-offs inherent in adaptive bitrate algorithms. While A2C delivers the best QoE, its high rebuffering time may make it less suitable for bandwidth constrained scenarios. Conversely, DDPG ensures a smooth playback experience but at the cost of lower QoE. While the findings do present great potential of RL within ABR steaming algorithms, they also show the importance of tailoring adaptive streaming algorithms to specific application requirements. Before these models are able to be truly useful, the polarization caused by over-focusing on one parameter to maximize QoE must be addressed.

There are still several directions that can be taken for future work on this project. First, the user QoE function can continue to be tuned in order to reduce the polarization of metrics that make up the score. While the A2C model in this experiment has shown that it is able to outperform the Dynamic and Bola QoE, it was due to always choosing high bitrates at the expense of rebuffering. Further modification of the QoE function in addition to the parameters that has great potential to reduce the polarity of metrics that make up QoE.

Now, this would not truly be an AI paper without the mention of increasing the data size. Due to the computational and time constraints of this project, less than 50% of the total dataset was utilized for training and testing the models. Increasing the dataset size and incorporating a broader range of trace conditions in future work could provide a more comprehensive evaluation of model performance under increasingly variable scenarios. Additionally, this experiment maintained a constant number of bitrate options and used the same video content across all traces. Future research could explore varying these parameters during training to enhance the models robustness and adaptability.

Another promising avenue for future work is the generation of ideal training episodes. By leveraging an oracle perspective with knowledge of future network conditions, it is possible to calculate the optimal bitrate selection at each time step to achieve the maximum possible QoE. Generating these optimal decisions for each trace and using them as a foundation for model training could significantly enhance performance, potentially leading to better overall results.

References

Blender. 2008. Big Buck Bunny.

Commission, F. C. 2016. Raw Data - Measuring Broadband America 2016.

Dobrian, F.; Sekar, V.; Awan, A.; Stoica, I.; Joseph, D.; Ganjam, A.; Zhan, J.; and Zhang, H. 2011. Understanding the impact of video quality on user engagement. *SIGCOMM Comput. Commun. Rev.*, 41(4): 362–373.

Lillicrap, T. P.; Hunt, J. J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; and Wierstra, D. 2019. Continuous control with deep reinforcement learning. arXiv:1509.02971.

Mnih, V.; Badia, A. P.; Mirza, M.; Graves, A.; Lillicrap, T. P.; Harley, T.; Silver, D.; and Kavukcuoglu, K. 2016. Asynchronous Methods for Deep Reinforcement Learning. arXiv:1602.01783.

Spiteri, K.; Sitaraman, R.; and Sparacio, D. 2018. From theory to practice: improving bitrate adaptation in the DASH reference player. In *Proceedings of the 9th ACM Multimedia Systems Conference*, MMSys '18, 123–137. New York, NY, USA: Association for Computing Machinery. ISBN 9781450351928.

Spiteri, K.; Urgaonkar, R.; and Sitaraman, R. K. 2020. BOLA: Near-Optimal Bitrate Adaptation for Online Videos. *IEEE/ACM Transactions on Networking*, 28(4): 1698–1711.

uMass. 2013. DATASET: HSDPA-bandwidth logs for mobile HTTP streaming scenarios.

van der Hooft, J.; Petrangeli, S.; Wauters, T.; Huysegems, R.; Alface, P. R.; Bostoen, T.; and De Turck, F. 2016. HTTP/2-Based Adaptive Streaming of HEVC Video Over 4G/LTE Networks. *IEEE Communications Letters*, 20(11): 2177–2180.

Wei, X.; Zhou, M.; Kwong, S.; Yuan, H.; Wang, S.; Zhu, G.; and Cao, J. 2021. Reinforcement learning-based QoE-oriented dynamic adaptive streaming framework. *Information Sciences*, 569: 786–803.

Yin, X.; Jindal, A.; Sekar, V.; and Sinopoli, B. 2015. A Control-Theoretic Approach for Dynamic Adaptive Video Streaming over HTTP. *SIGCOMM Comput. Commun. Rev.*, 45(4): 325–338.