TIYUNTSONG: A SELF-PLAY REINFORCEMENT LEARNING APPROACH FOR ABR VIDEO STREAMING

Tianchi Huang, Xin Yao, Chenglei Wu, Rui-Xiao Zhang, Zhengyuan Pang, Lifeng Sun

Dept. of Computer Science and Technology, Tsinghua University, Beijing, China
{htc17,yaox16,wucl18,zhangrx17,pangzy12}@mails.tsinghua.edu.cn, sunlf@tsinghua.edu.cn

ABSTRACT

Existing reinforcement learning (RL)-based adaptive bitrate (ABR) approaches outperform the previous fixed control rules based methods by improving the Quality of Experience (QoE) score, as the QoE metric can hardly provide clear guidance for optimization, finally resulting in the unexpected strategies. In this paper, we propose *Tiyuntsong*, a self-play reinforcement learning approach with generative adversarial network (GAN)-based method for ABR video streaming. Tiyuntsong learns strategies automatically by training two agents who are competing against each other. Note that the competition results are determined by a set of rules rather than a numerical QoE score that allows clearer optimization objectives. Meanwhile, we propose GAN Enhancement Module to extract hidden features from the past status for preserving the information without the limitations of sequence lengths. Using testbed experiments, we show that the utilization of GAN significantly improves the Tiyuntsong's performance. By comparing the performance of ABRs, we observe that Tiyuntsong also betters existing ABR algorithms in the underlying metrics.

***Index Terms*—** Adaptive Bitrate Streaming, Self-play Reinforcement learning

1. INTRODUCTION

Recent years have witnessed a rapid growth of online video streaming applications and services [1]. To achieve smooth video playback under various network conditions, modern client-side video player adopts ABR algorithm to dynamically determine the bitrate of next video chunk to download to achieve high QoE scores including high video bitrate, low rebuffering, etc. Most of the approaches, such as throughput-based[2], buffer-based[3] and mixed schemes[4, 5] employ fixed control rules which determine future video bitrates via carefully tuned strategies and thresholds. However, these approaches are often designed with strong assumptions of the real-world network conditions and heavily rely on the fine-tuned parameters, which results in sensitivities to network conditions and unexpected performances (§2.1). To address these problems, researchers [6] have proposed to leverage reinforcement learning (RL) methods to *learn* an algorithm from scratch without any network presumptions. In particular, the state-of-the-art deep reinforcement learning (DRL) scheme Pensieve [6] outperforms existing ABRs in some settings [7]. These work tries to optimize a neural network towards a better QoE score, in which the fine-tuned parameters have significant impacts on the performance. However, through the experiment, we observe that RL-based method often obtains high QoE scores via some tricks due to the lack of guidance for optimization in QoE (§2.2). As a result, despite its abilities to obtain higher numerical QoE scores, such training schemes may generate a strategy that doesn't meet the basic rules of the ABR algorithm.

Our key idea is to regard the reward as a rule instead of QoE metrics (§2.3). The rule (§4.1) is allowed to be constructed by any methods, such as a logistic and AI method, aiming to identify the better one from two candidates. Unlike previous work, the rule will highlight the priority of optimization to avoid occurring unexpected strategies. The novel idea requires a new RL framework to match it. Thus, we propose *Tiyuntsong*[1], a self-play RL method with GAN for ABR video streaming (§3). Tiyuntsong trains two agents simultaneously for generating a well-performed ABR algorithm under different network conditions. In details (§3.3), Tiyuntsong first uses two agents to provide the video streaming services on the same network condition and video content respectively. Next, it leverages the rule to determine who is the winner. Finally, it assigns the reward of each agent as {*win:1, lose:0*} and updates the two agents' gradients. In brief, Tiyuntsong approaches a Nash equilibrium via the self-play method, whereas traditional RL methods diverge.

Besides, we further present *GAN Enhancement Module* (§3.2), a GAN-based method to extract hidden features from the past status that facilitate Tiyuntsong to store the information without the limitation of sequence length. During the training process, the model generates future hidden features based on current state and hidden features, and then estimates the probability of whether the hidden feature comes

[1]Tinyuntsong: Also named as *Cloud Ascending Ladder*, a qinggong skill in the Chinese wuxia novel *The Heaven Sword and Dragon Saber* by Jin Yong. The skill enables the user to travel at high speeds and leap to extreme heights by stepping one foot on the other one.

from the winning sample or not.

In the rest of our paper (§4), first, we collect a large corpus of network traces from alternative public datasets for training and validating. Next, we leverage Elo-Rating [8], a classic rating-based system to compute the performance of Tiyuntsong via win rate (§4.1). Finally, using a testbed experiment (§4.2), we first discuss Tiyuntsong's neural network architecture (§4.2.1). After that, we prove the importance of GAN Enhancement Module (§4.2.2). In all considered scenarios, Tiyuntsong betters existing ABR approaches in both win rate and underlying metrics of ABR including bitrate and rebuffering as well as smoothness (§4.2.3).

To sum up, our contributions are as follows:

▷ We figure out the weakness of RL-based ABR algorithms and suggest a novel sight to redefine the reward metric for them: leverage logistic rules instead of QoE metrics.

▷ To the best of our knowledge, we are the first to use self-play RL method to tackle the ABR video streaming problem. Results indicate that Tiyuntsong not only avoids deviating from the fundamental rule but also betters recent work.

▷ Traditional fusion of RL and GAN method [9] pays more attention to improving the efficiency of imitation rather than preserving the useful information as described in this paper. In brief, we propose a novel perspective for the application of GAN, which also yields a reliable and effective result.

2. RELATED WORK AND MOTIVATION

2.1. ABR Algorithms Overview

Client-based ABR algorithms are mainly categorized into four types [10]: throughput-based, buffer-based, mixed and RL-based. PANDA [2] predicts the future throughput for eliminating the ON-OFF steady issue. However, due to the current lack of throughput estimation method, these approaches still result in poor ABR performance. Thus, many approaches are designed to select the proper bitrates based on playback buffer size observed. e.g., BOLA [3] turns the ABR problem into a utility maximization problem and solve it by using the Lyapunov function. Nevertheless, the buffer-based approach fails to tackle the long-term bandwidth fluctuation problem. Then, to tackle the challenge, mixed approaches, such as MPC [4] and DynamicDASH [5], is proposed to select bitrate for next chunk by adjusting its throughput discount factor based on past prediction errors and predicting its playback buffer size. Note that such model-based approaches require careful tuning, because they rely on parameters that are quite sensitive to network conditions, which results in the poor performance in unexpected network environments. To address these issues, several attempts [6] have been made to optimize ABR algorithms based on RL method due to the difficulty of tuning mixed approaches for handling different network conditions.

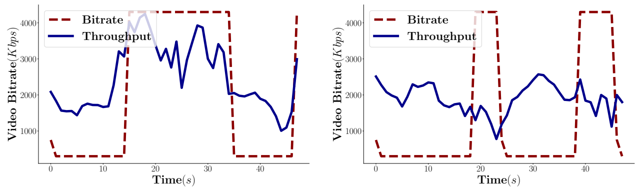

Fig. 1. This group of figures shows the drawback of traditional RL-based ABR methods: After trained Pensieve with default parameter settings about seven days, we observed that: 1) Its policy was simple and effective; 2) It still maintained a high QoE score. However, it's clear that the proposed method deviates from the basic rules of ABR.

2.2. The Trap of Traditional RL-based Method

However, traditional RL-based methods still have their drawbacks. Considering the ABR process as a Markov Decision Process (MDP), in recent years, many schemes have been proposed to learn ABR algorithms via RL method [11, 6]. Despite the outstanding performances that RL-based ABR algorithms achieve, these schemes suffer from a key limitation: They optimize their neural network via enhancing QoE scores. However, achieving a high QoE score doesn't necessarily equal to generate an exemplary algorithm. For example, the experimental results of state-of-the-art RL-based scheme Pensieve[2] is illustrated in Figure 1. While Pensieve converges, its policy can be generalized as

Fetching lowest bitrates till the buffer has enough space and time to fetch highest bitrates.

That is because though RL methods will successfully train the model under the case which only needs to optimize one metric (e.g., to play higher scores for Atari games), these schemes lack the abilities to tackle the problem affected by multiple factors directly. For example, in ABR problems, recent work [3, 4, 6] leverages reward functions (e.g., QoE metrics) with a weighted sum of several underlying metrics to take the place of the multi-factor optimization (Several metrics must be optimized together). Hence, QoE driven RL-based approaches have the abilities to obtain a relative good numerical reward, but they may also provide users with unexpected and unstable viewing experience. This problem imposes critical challenges to RL-based ABR algorithm.

2.3. Self-play Method

AlphaZero [12], a scientific breakthrough in AI, is trained solely based on self-play RL and periodically matched against several games. To tackle the traditional RL methods' problem, we thereby consider following AlphaZero to train the algorithm via self-play RL. However, static games with incomplete information (e.g., training ABR with two agents) are much dissimilar and more complicated than dynamic games

[2]https://github.com/hongzimao/pensieve

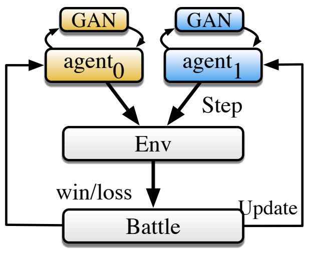

Fig. 2. Tiyuntsong overview

with complete information such as Go. Thus, we meet new challenges: *How to design a proper model and how to define a suitable reward representation for ABR?*

3. TIYUNTSONG'S MECHANISM

In this section, we provide the design steps and the training methodology of Tiyuntsong [3]. As stated before (§2.2), conventional RL is not a suitable scheme to solve the complex reward function problem in which its reward is computed as a linear combination of multiple factors. We, therefore, suggest a novel sight to describe the reward function: Only to represent the reward as *win* or *loss* instead of an actual reward score. Following this sight, we propose Tiyuntsong, a self-play with RL method which learns an algorithm automatically based on the *win or loss signal* only. As illustrated in Figure 2, two agents compete for each other in the same environment and then update their network based on the competitive result.

3.1. The Design of Agent

We first initialize two agents $\mathbf{A_0}$ and $\mathbf{A_1}$. It is worth noting that Tiyuntsong's neural network architecture is quite different from the common RL's representation due to the distinctiveness of the ABR task. The rest of the details are described as follows.

State: Tiyuntsong's learning agent takes the input state of time-slot t $s_t = \{T, d, q, r, b, \mathbb{S}, h\}$ into neural network, where T means the past throughput measured by a client for past k sequence; d represents the time for downloading past k sequence; q is the previous video bitrate selected of past k sequence; r is the remaining video playback time; b is a buffer length used by the client; $\mathbb{S}$ is a vector that represents the video sizes of the next video chunk. The last one h is a vector that reflects that extra features of the past, and it is generated by the GAN Enhancement Module (See §3.2).

Action: The action space is discrete. The output of the policy network is defined as a probability distribution: $f(s_t, a_t)$, meaning the probability of selection action a_t being in state s_t. The action a_t is an n-dims vector, which represents the candidate of video bitrate for the next chunk.

[3]https://github.com/thu-media/tiyuntsong

Reward: Our reward is defined as a result: $r \in \{0, 1\}$ judged by `Rule`. During the training process, we use `Rule` to compute the win rate of two agents for each epoch, and in the result "0" means loss and "1" represents victory. We notice that `Rule` can be represented as not only a human-made logistic algorithm but also a neural network model generalized by AI. The details of `Rule` are described in §4.1. Based on the results calculated, we can estimate the win rate w_i for each agent $\mathbf{A_i}$. We further utilize Elo Ratings to estimate the instant performance via win rate (§4.1).

3.2. GAN Enhancement Module

In recent work [13, 6], the lifetime of each bitrate decision has been modeled as an MDP (Markov Decision Process), meaning that *action* is only related to the status of the target rather than relying on the prior states. However, this assumption lacks evidence. [13] only illustrates that throughput factors can be efficiently captured by Hidden-Markov-Model (HMM). Still, in [6], the authors also consider different numbers of past throughout measurements to represent *state*. In general, previous work leverages a past k steps status observed to reckon the status of the target in MDP, and the limitation of sequence length leads to missing crucial information of the past, such as the maximum and minimum values of the observed throughput.

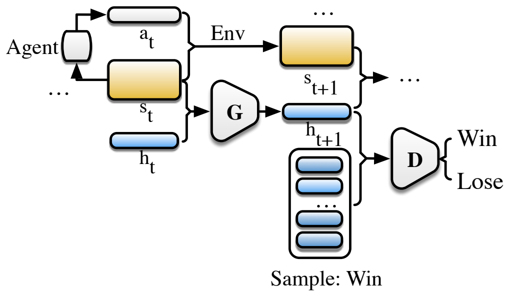

Fig. 3. GAN Enhancement Module

Thus, we present *GAN Enhancement Module* to automatically generate the hidden features from the past to break the limitation of sequence length. As illustrated in Figure 3, the module consists of a generator $\mathbf{G}$ and a discriminator $\mathbf{D}$, where $\mathbf{G}$ is a function represented by several fully connected layers using *leaky RelU* with parameters θ_g, and $\mathbf{D}$ is also a function represented by multilayers with parameters θ_d. For each step t, $\mathbf{G}$ is used for generating next hidden features h_t according to the state s_{t-1} and hidden feature h_{t-1}, and $\mathbf{D}$ outputs a single scalar $p_t \in [0, 1)$ to estimate the probability that h_t belongs to the historical winning samples. Inspired by LSGAN [14], we first update $\mathbf{D}$ by descending its gradient according to $\mathbf{L}_d$ (Eq. 1). We then update $\mathbf{G}$ by descending its gradient via $\mathbf{L}_g$ (Eq. 2). Here $\mathbf{w}$ means the winning samples defined as $\mathbf{w} = \{h_0, h_1, \cdots, h_k\}$, where reward $r_k = 1$.

$$\min_{\mathbf{D}} \mathbf{L}_d = \frac{1}{2} E_{x \sim p_w(x)}[(\mathbf{D}(x) - 1)^2] + \frac{1}{2} E_{t \sim p_g(t)}[(\mathbf{D}(\mathbf{G}(s_{t-1}, h_{t-1})))^2] \quad (1)$$

$$\min_{\mathbf{G}} \mathbf{L}_g = \frac{1}{2} E_{t \sim p_g(t)}[(\mathbf{D}(\mathbf{G}(s_{t-1}, h_{t-1})) - 1)^2] \quad (2)$$

3.3. Training Methodology

We now start to discuss how to train Tiyuntsong. In our work, we use the actor-critic method as the fundamental algorithm of Tiyuntsong. Each agent is composed of a policy network and a value network. The key thought of the policy gradient algorithm is to update the parameter in the direction of increasing the accumulated reward. The gradient of the accumulated reward with respect to policy parameter θ can be written as

$$\nabla E_{\pi_\theta}[\sum_{t=0}^{\infty} \gamma^t r_t] = E_{\pi_\theta}[\nabla_\theta \log_{\pi_\theta}(s, a) A^{\pi_\theta}(s, a)]. \quad (3)$$

We can use $E_\theta[\nabla_\theta log \pi_\theta(s, a) A^{\pi_\theta}(s, a)]$ as its unbiased form, where $A(s_t, a_t)$ is called the advantage of action a_t in state s_t which satisfies the following equality: $A(a_t, s_t) = Q(a_t, s_t) - V(s_t)$, where $V(s_t)$ represents the estimate of the value function of state s_t and $Q(a_t, s_t)$ is the value of taking certain action at state s_t. Next, we consider to use n-step Q-learning for optimizing the value network. The value network will be updated as

$$\theta_v \leftarrow \theta_v - \alpha_v \sum_t \nabla_{\theta_v} (\underbrace{r_t + \gamma V(s_{t+1}|\theta_v)}_{Q(a_t, s_t)} - V(s_t|\theta_v))^2. \quad (4)$$

Here $V(s_t|\theta_v)$ is the estimation of $V(s_t)$; The direction of changing parameter θ_v is the negative gradient of it; α_v is the learning rate for the value network. We also add the entropy of policy in the object of policy network, which can effectively discourage the network to converge to sub-optimal policies. Thus, the update of θ will be written as

$$\theta \leftarrow \theta + \alpha_p \sum_t \nabla_\theta \log_{\pi_\theta}(s_t, a_t) A(s_t, a_t) + \beta \nabla_\theta H(\pi_\theta(\cdot|s_t)), \quad (5)$$

where $H(\cdot)$ is the entropy of the policy. After convergence, the value network will be abandoned, and we only use policy network to make decisions; α_p is a learning rate function; β is a hyper-parameter regarded as the weight of exploration. For each epoch $i (i > 0)$, the parameters α_{p_i} and α_{v_i} can be computed by the equalization as follows:

$$\alpha_{p_i}, \alpha_{v_i} = \begin{cases} (\alpha_{p_0}, \alpha_{v_0})(w_i \log w_i + 2.0) & w_i < 0.5 \\ -(\alpha_{p_0}, \alpha_{v_0}) w_i \log w_i & w_i \geq 0.5, \end{cases} \quad (6)$$

in which w_i is the win rate for each training epoch i; α_{p_0} and α_{v_0} are initialized hyper-parameters which control the overall learning rate of policy network and value network. Dynamic learning rate can effectively avoid a considerable gap between the two agents.

4. EVALUATION

4.1. Experimental Setup

Datasets We collect about 2,300 network traces from different public datasets for training and evaluating Tiyuntsong. The details of our network traces are composed of Norway [15], Synthetic Network Traces [6], Belgium [16], FCC [6], and Oboe [7].

Table 1. The `rule` Used In The Experiment

(a)

`Rule`	$b_0 > b_1$	$b_0 = b_1$	$b_0 < b_1$
$r_0 > r_1$	Table 1(b)	**1**	**1**
$r_0 = r_1$	**0**	Table 1(c)	**1**
$r_0 < r_1$	**0**	**0**	Table 1(b)

(b)

$r_0/b_0 > r_1/b_1$	**1**
$r_0/b_0 = r_1/b_1$	**0**
$r_0/b_0 < r_1/b_1$	Table 1(c)

(c)

$s_0 > s_1$	**1**
$s_0 = s_1$	**0/1**
$s_0 < s_1$	**0**

The Design of `Rule` A good ABR algorithm mainly consists of three underlying metrics [5]:

▷ **Bitrate:** To play the video at the highest sustainable quality, such as bitrate and video quality.

▷ **Rebuffering:** To avoid rebuffering events that occur due to the empty client buffer.

▷ **Smoothness:** Keep the bitrate in little change during the entire session.

To this end, we implement an intuitive logistic rule[4] for evaluation based on their priority (See in Table 1), in which b_i denotes total bitrate, r_i is total rebuffer time and s_i represents total bitrate change for each agent $i \in \{0, 1\}$.

Metrics We leverage Elo Ratings [8], a traditional method for calculating the relative performance of players in zero-sum games, to evaluate Elo Ratings based on win rate. We first select several previously proposed approaches and test their performance respectively under the same environment. Next, we use `rule` to estimate their win rate. Finally, we compute the Elo rating for these approaches. In our work, these scores are defined as *baselines*. For each epoch, the

[4]We repeat that `Rule` is allowed to design in any way, i.e., logistic or AI methods, etc. In this paper, due to the space limitations, we only give an intuitive rule for evaluating Tiyuntsong because too many statements about the rules will be badly miscast here.

Arch.	Elo	Timespan(it/s)
FC	1033	**1.28**
LSTM	1057	0.77
2D-CNN	1040	1.16
1D-CNN	**1094**	1.04
Constrained	977	-
Throughput-Rule	1023	-

Table 2. Comparing performance (Elo Ratings) of Tiyuntsong with different neural network architectures including Fully Connected, 2D-CNN, 1D-CNN and LSTM. Results are evaluated under same network traces and video description in 50 steps.

agent compares the result with baselines and then computes the Elo rating through the win rate. In this experiment, we set hyper-parameter $K = 10$ and initialized rating $I = 1000$ for the Elo system. More details are described in our repo..

Testbed Setup We utilize Sabre [5], a state-of-the-art open-sourced simulation environment for ABR algorithms, to precisely emulate the ABR's process in an offline environment. Sabre is a Python tool that can quickly evaluate ABR algorithms in an emulated environment similar to real production players. For each step t, the agent uses the Sabre environment to simulate the entire session with given video descriptions and network traces.

4.2. Experiments and Results

4.2.1. Tiyuntsong with Different Architectures

In this experiment, we compare Tiyuntsong's network architecture to the following network architectures which collectively represent the architecture candidates:

Fully Connected: $\mathtt{FC}^1_{64} \rightarrow \mathtt{FC}^2_{128} \rightarrow \mathtt{FC}^3_{64}$

LSTM: $\mathtt{LSTM}^1_{64} \rightarrow \mathtt{LSTM}^2_{64} \rightarrow \mathtt{SELF\text{-}ATTENTION}^1_{64}$

2D-CNN: $\mathtt{CONV2D}^1_{64} \rightarrow \mathtt{MAXPOOL}^1_2 \rightarrow \mathtt{CONV2D}^2_{64} \rightarrow \mathtt{MAXPOOL}^2_2 \rightarrow \mathtt{FC}^1_{64}$

1D-CNN*: $\mathtt{CONV1D}^{1\cdots6}_{64} \rightarrow \mathtt{MERGE}^1 \rightarrow \mathtt{FC}^1_{64}$

We train and test under Sabre with the same network traces and video descriptions. In this experiment, we set $\gamma = 0.99$, $\beta = 0.02$, $step = 50$ for only testing their performance instead of convergence. We report the result in Figure 9, where 1D-CNN is the Tiyuntsong's network architecture. The obtained results indicate that 1D-CNN neural network architecture succeeds in improving the Elo Ratings, with improvements in average Elo Ratings of 37 - 61. We also observe that there is no obvious difference between these architectures regarding operational efficiency.

4.2.2. Tiyuntsong vs. Tiyuntsong without GAN

In this part, we design an experiment to confirm whether the GAN Enhancement Module is effective or not. We set $step =$ 50 and compare Tiyuntsong-GAN with Tiyuntsong without using GAN Enhancement Module on the same network traces, and use two existing approaches: constrained and throughput rule as baselines. The experimental result is illustrated in Figure 4. As expected, we observe that Tiyuntsong-GAN outperforms Tiyuntsong with improvements in average Elo Ratings of 13.3% after 50 steps.

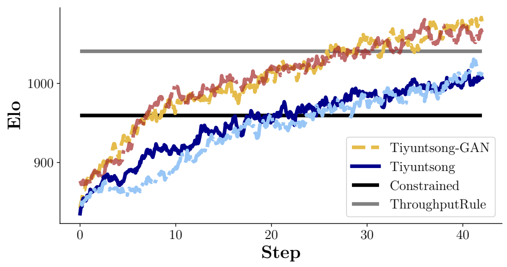

Fig. 4. Comparing Tiyuntsong-GAN with Tiyuntsong on the same network traces. Results are evaluated in Elo Ratings based on previous approaches as baselines.

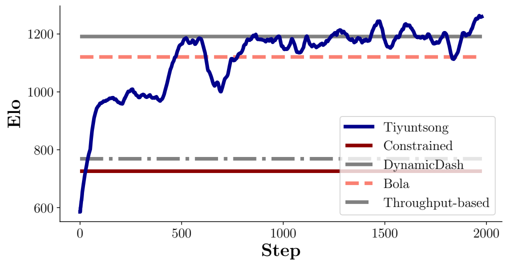

Fig. 5. The curve of training Tiyuntsong for 2,000 steps. Elo-ratings are computed from `Rule` between different ABR algorithms including Constrained, Throughput-Based, Bola and DynamicDASH.

4.2.3. Tiyuntsong vs. Existing ABR Approaches

In this experiment, we aim to evaluate the Elo Ratings of several existing ABR algorithms including BOLA, DynamicDASH, Throughput-based, Constrained, and Pensieve (QoE-lin). BOLA and DynamicDASH have been implemented in [5], and we use the harmonic mean of past five throughput measured to present the throughput-based rule. Moreover, we denote the constrained rule to select the intermediate chunk of the next video chunks and train a model via Pensieve optimized by `QoE-lin` [4]. We train Tiyuntsong about 2,000 epochs on the network traces datasets. We use 80% dataset for training, 20% for validating and leverage Oboe dataset for testing. Figure 5 shows the performance of training Tiyuntsong for 2,000 steps, we observe that Tiyuntsong performs better than the existing approaches after 1,800 steps. We also report Tiyuntsong's win rate and the CDF distribution of three underlying metrics in Figure 6. Compared to Dynam-

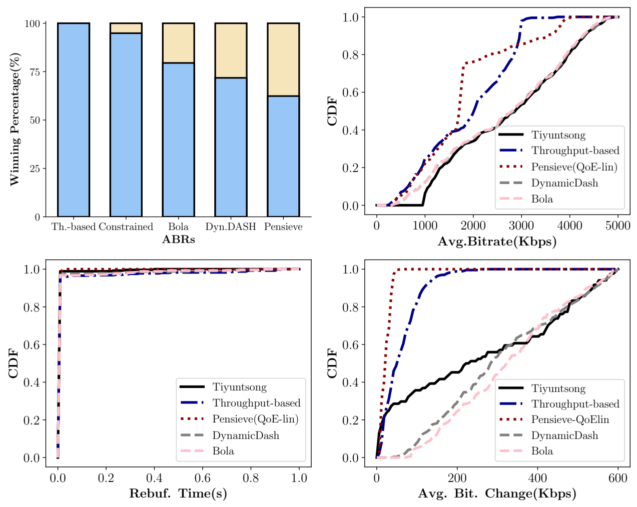

Fig. 6. Comparing Tiyuntsong with existing ABR approaches on the same network traces. Results is shown with the win rate and the distribution of average bitrate, rebuffering time and average bitrate change for the approaches. Results show that Tiyuntsong wins existing approaches, with the win rate of 62% to 100%.

icDash, Tiyuntsong improves the average bitrate by 3.19%, decreases the average rebuffering time by 4.92%, and reduces the 95th percentile average bitrate change by 16.47% respectively. As expected, we also observe that Pensieve does reach an overwhelming advantage on the QoE metric but fails to perform well under some underlying metrics such as average bitrate, and it also proves our motivation of this work.

5. CONCLUSIONS AND FUTURE WORK

We propose Tiyuntsong, self-play RL approach to select bitrates for next video chunk. Unlike previously proposed approaches, Tiyuntsong uses two agents to compete against each other for automatically generating a better ABR algorithm. Experimental results prove that Tiyuntsong has achieved the state-of-the-art ABR algorithm via self-play. Additional research may focus not only to accelerate the training process but also to extend our work to solve the general incomplete information game problem.

Acknowledgements

We thank the anonymous reviewers for their valuable feedback. I also thank my wife Yuyan Chen for her great supports and brave ideas. The work is supported by the National Key R&D Program of China (2018YFB1003703)NSFC under Grant 61521002, Beijing Key Lab of Networked Multimedia, Alibaba-Tsinghua Joint Project (20172001689).

6. REFERENCES

[1] Cisco, "Cisco visual networking index: Forecast and methodology, 2016-2021," 2017.

[2] Zhi Li, Xiaoqing Zhu, Joshua Gahm, Rong Pan, Hao Hu, Ali C Begen, and David Oran, "Probe and adapt: Rate adaptation for http video streaming at scale," *IEEE JASC*, vol. 32, no. 4, pp. 719–733, 2014.

[3] Kevin Spiteri, Rahul Urgaonkar, and Ramesh K Sitaraman, "Bola: Near-optimal bitrate adaptation for online videos," in *INFOCOM 2016, IEEE*. IEEE, 2016, pp. 1–9.

[4] Xiaoqi Yin, Abhishek Jindal, Vyas Sekar, and Bruno Sinopoli, "A control-theoretic approach for dynamic adaptive video streaming over http," in *SIGCOMM 2015*. ACM, 2015, pp. 325–338.

[5] Kevin Spiteri, Ramesh Sitaraman, and Daniel Sparacio, "From theory to practice: improving bitrate adaptation in the dash reference player," in *MMSys 2018*. ACM, 2018, pp. 123–137.

[6] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh, "Neural adaptive video streaming with pensieve," in *SIGCOMM 2017*. ACM, 2017, pp. 197–210.

[7] Zahaib Akhtar, Yun Seong Nam, Ramesh Govindan, et al., "Oboe: auto-tuning video abr algorithms to network conditions," in *SIGCOMM 2018*. ACM, 2018, pp. 44–58.

[8] A.E. Elo, *The rating of chessplayers, past and present*, Arco Pub., 1978.

[9] Thang Doan, Bogdan Mazoure, and Clare Lyle, "Gan q-learning," *arXiv preprint arXiv:1805.04874*, 2018.

[10] Abdelhak Bentaleb, Bayan Taani, Ali C Begen, Christian Timmerer, and Roger Zimmermann, "A survey on bitrate adaptation schemes for streaming media over http," *IEEE Communications Surveys & Tutorials*, 2018.

[11] Federico Chiariotti, Stefano D'Aronco, Laura Toni, and Pascal Frossard, "Online learning adaptation strategy for dash clients," in *Proceedings of the 7th International Conference on Multimedia Systems*. ACM, 2016, p. 8.

[12] David Silver, Thomas Hubert, Julian Schrittwieser, et al., "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," *arXiv preprint arXiv:1712.01815*, 2017.

[13] Yi Sun, Xiaoqi Yin, Junchen Jiang, et al., "Cs2p: Improving video bitrate selection and adaptation with data-driven throughput prediction," in *SIGCOMM 2016*. ACM, 2016, pp. 272–285.

[14] Xudong Mao, Qing Li, Haoran Xie, et al., "Least squares generative adversarial networks," in *ICCV, 2017*. IEEE, 2017, pp. 2813–2821.

[15] Haakon Riiser, Paul Vigmostad, Carsten Griwodz, and Pål Halvorsen, "Commute path bandwidth traces from 3g networks: analysis and applications," in *Proceedings of the 4th ACM Multimedia Systems Conference*. ACM, 2013, pp. 114–118.

[16] Jeroen van der Hooft, Stefano Petrangeli, Tim Wauters, et al., "Http/2-based adaptive streaming of hevc video over 4g/lte networks," *IEEE Communications Letters*, vol. 20, no. 11, pp. 2177–2180, 2016.

[17] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, 2014, pp. 2672–2680.

[18] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville, "Improved training of wasserstein gans," in *Advances in Neural Information Processing Systems*, 2017, pp. 5767–5777.

[19] Ming-Yu Liu and Oncel Tuzel, "Coupled generative adversarial networks," in *Advances in Neural Information Processing Systems 29*, D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, Eds., pp. 469–477. Curran Associates, Inc., 2016.

[20] Eric Tzeng, Judy Hoffman, Kate Saenko, and Trevor Darrell, "Adversarial discriminative domain adaptation," in *Computer Vision and Pattern Recognition (CVPR)*, 2017, vol. 1, p. 4.

[21] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al., "Tensorflow: A system for large-scale machine learning.," in *OSDI*, 2016, vol. 16, pp. 265–283.

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

[23] Tijmen Tieleman and Geoffrey Hinton, "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude," *COURSERA: Neural networks for machine learning*, vol. 4, no. 2, pp. 26–31, 2012.

[24] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International Conference on Machine Learning*, 2016, pp. 1928–1937.

A. APPENDIX

This supplementary material details the principle of Tiyuntsong[5]. Due to the length of the supplemental material, we list a content to facilitate the selection of interested parts for review. Although these contents have **NOT** appeared in the main text, we believe that they will help the reviewer get a thorough understanding of Tiyuntsong.

- Related work including adversarial learning (§B.1) and ABR's background (§B.2);
- Tiyuntsong's network architecture (§C.1), training procedure (§C.2);
- Tiyuntsong's training time (§D.1), another experiment and result for evaluating Tiyuntsong's performance under different network architecture candidates (§D.2);
- Additional discussions: The relationship between traditional QoE functions and Rules (§E.1), Why these network traces are selected? (§E.2)

[5]In memory of Jin Yong (1924 - 2018).

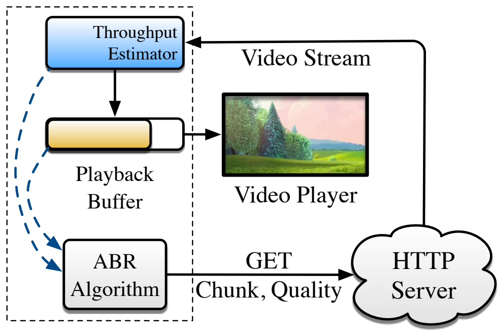

Fig. 7. An overview of ABR video streaming.

B. RELATED WORK

B.1. Adversarial Learning

Since GAN first proposed [17], the adversarial discriminative learning method has been widely used in the various fields. The original GAN model is short of the loss function. Thus, many approaches, such as LSGAN [14] and WGAN-gp [18], extend GAN's training methodology with strong theoretical proof. Moreover, adversarial learning has also been applied to extract features. For example, CoGAN [19] uses GAN to solve the domain transfer issue, and ADDA [20] proposes a GAN-based generalized framework for domain adaptation.

B.2. Background on ABR

Due to the rapid development of network services, watching video streaming online has become an upcoming trend. Today, adaptive video streaming, such as HLS (HTTP Live Streaming) and DASH, an algorithm that dynamically selects video bitrates via network conditions and client's buffer occupancy, is the predominant form of video delivery. The traditional video streaming architecture is shown in Figure 7, which consists of a video player client with a constrained buffer length and an HTTP-Server or Content Delivery Network (CDN). The video player client decodes and renders video frames from the playback buffer. Once the streaming service starts, the client fetches the video chunk from the HTTP Server or CDN orderly by an ABR algorithm, and, in the meanwhile, the ABR algorithm, implemented on the client side, determines the next chunk N and next chunk video quality Q_N via throughput estimation and current buffer utilization.After finished to play the video, several metrics, such as total bitrate b, total re-buffering time r and total bitrate change s will be summarized as a QoE metric to evaluate the performance. Thus, achieving a high QoE score for video streaming has become a major challenge for ABR algorithms.

To overcome this challenge, most traditional ABR algorithms leverage time-series prediction (throughput-based) or automation control method (buffer-based) to make decisions for the next chunk. Moreover, [6] suggests that traditional fixed control rules methods require careful tuning and will achieve bad performances in the circumstance which is different from the assumption. As a result, traditional ABR algorithms perform well in pre-assumption and specific network conditions but hard to keep its performance in various network environments.

C. TIYUNTSONG'S DETAILS

C.1. Architecture and Implementation Details

We use TensorFlow [21] to implement Tiyuntsong[6]. As demonstrated in Figure 8, Tiyuntsong is composed of five neural network architectures as follows.

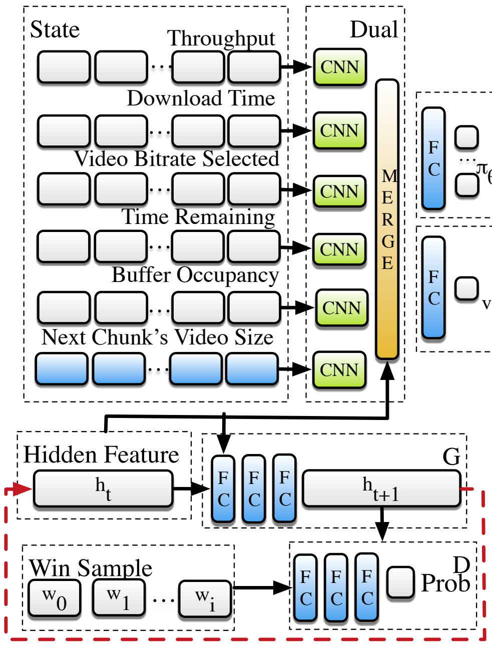

Fig. 8. Tiyuntsong's network architecture.

Dual network: We set past sequence length $k = 10$. Features are extracted from the input state via a feature extraction layer. For each feature in the input state, it's passed through a conv-1d layer with 64 filters and the kernel size of 1×3. Meanwhile, we use *ReLU* function as the activation function after each layer. Finally, the feature maps are concatenated as a tensor.

Policy network & Value network: Both policy network and value network are performed behind the Dual network. We use a fully connected layer with 64 neurons and active function *ReLU* to represent them. The output of each network is n-dim vector and a single scalar respectively. In this work, we set $\gamma = 0.6$, $\beta = 0.01$, the learning rate for policy network $\alpha_0 = 10^{-4}$, and the learning rate for value network $\alpha_v = 10^{-3}$. In this experiment, we use Adam optimizer [22] with default parameters to optimize these neural networks.

Generative network & Discriminator network: Like previous work, the generative network and discriminator are composed of fully connected layer `FC` and batch normalization layer `BN`. The generative network architecture is described as $\texttt{FC}^1_{64} \rightarrow \texttt{BN}^1 \rightarrow \texttt{FC}^2_{32} \rightarrow \texttt{BN}^2 \rightarrow \texttt{FC}^3_{16}$, and the discriminator network is listed as $\texttt{FC}^1_{64} \rightarrow \texttt{BN}^1 \rightarrow \texttt{FC}^2_{32} \rightarrow \texttt{BN}^2 \rightarrow \texttt{FC}^3_{1}$. Meanwhile, we use *Leaky ReLU* as the active function and set learning rate for the generative network and discriminator network $\alpha_G = \alpha_D = 10^{-4}$, hidden feature size $size_{h_t} = 16$. Referring to the recommendations in LSGAN [14], we use RMSProp optimizer [23] to update their gradients.

C.2. Tiyuntsong's Training Procedure

See details in Algorithm 1.

Algorithm 1 Tiyuntsong's Overall Training Procedure

Require: The ABR environment `Env` to measure total bitrate, total rebuffer time, and total bitrate change with given video samples $\mathbf{V}$ and network traces $\mathbf{T}$; Two agents with GAN Enhance Module $\{\mathbf{A}_0; \mathbf{G}_0, \mathbf{D}_0\}$ and $\{\mathbf{A}_1; \mathbf{G}_1, \mathbf{D}_1\}$; A rule for estimating reward `Rule`;.

```
1:  procedure TRAINING(Env, $\mathbf{T}$, $\mathbf{V}$, $\mathbf{A}_0$, $\mathbf{A}_1$)
2:      Initialize the parameters $\theta$ of $\theta_{\mathbf{A}_0}$ and $\theta_{\mathbf{A}_1}$ with random weights respectively.
3:      repeat                                        ▷ Epoch ← Epoch + 1
4:          Sample trace, video from $(\mathbf{T}, \mathbf{V})$;
5:          for $(t, v) \in$ (trace, video) do
6:              $(s_0, a_0) \leftarrow$ Env$(\mathbf{A}_0, t, v)$;
7:              $(s_1, a_1) \leftarrow$ Env$(\mathbf{A}_1, t, v)$;
8:          end for
9:          Compute reward $\mathbf{R} \in \{r_0, r_1\} \leftarrow$ Rule$(s_i, a_i)$;
10:         Estimate winning percentage $\mathbf{w} \in \{w_0, w_1\}$ from $\mathbf{R}$;
11:         for $i \in \{0, 1\}$ do
12:             Get winning samples $\mathbf{A}_{i_w}$;
13:             Update $\mathbf{D}_i$ and $\mathbf{G}_i$ with (1) and (2) using $(s_i, a_i, r_i, \mathbf{A}_{i_w})$;
14:             Update policy with (4) and (5) using $(s_i, a_i, r_i, w_i)$;
15:         end for
16:     until Converged
17: end procedure
```

C.3. Tiyuntsong meets Parallel Training

During the training process, we observe that the training progress is inefficient while using a single process. In-

[6][Online]Available: `https://github.com/`**`anonymous`**

Architecture	Elo	Timespan(it/s)
FC	1033	**1.28**
LSTM	1057	0.77
2D-CNN	1040	1.16
1D-CNN	**1094**	1.04
Constrained	977	-
Throughput-Rule	1023	-

Table 3. Comparing performance (Elo ratings) of Tiyuntsong with different neural network architectures including Fully Connected, 2D-CNN, 1D-CNN and LSTM. Results are evaluated under same network traces and video description in 50 steps.

spired by the multi-agent training method [24], we modify Tiyuntsong's training in the single agent as training in multi-agents. Multi-agents training consists of two parts, a central agent and a group of forwarding propagation agents. The forward propagation agents only decide with both policy and critic via state inputs and neural network model received by the central agent for each step; then it sends the n-dim vector containing $\{state, action, reward, gan\}$ to the central agent. The central agent uses the actor-critic algorithm to compute gradient and then updates its neural network model. Finally, the central agent pushes the updated network parameters to each forward propagation agent. Note that this can happen asynchronously among all agents, for instance, there is no locking between agents. By default, Tiyuntsong with multiple training uses 12 forward propagation agents and one central agent;

D. ADDITIONAL EVALUATIONS

Instead of original paper, We still evaluate Tiyuntsong under various neural network architectures. Our results answer the questions: How long will Tiyuntsong converge? What's the best neural network architecture for Tiyuntsong?

D.1. Tiyuntsong's Training Time

Tiyuntsong trains itself via endless competition, so the longer Tiyuntsong trains, the better it performs. In this paper, we stop training Tiyuntsong on i7-4790k CPU in 4 cores till its Elo-rating exceeds previous approaches. (Unlike traditional CV work, AI in networking requires a small model which can obtain high performance in low costs, so training on CPU is feasible). The training time lasts about 1.5 days. We observe that Tiyuntsong outperforms previously proposed approaches in 40mins, 2hrs, 13hrs, and 33hrs respectively. We will focus on accelerating the training process as is described in future work.

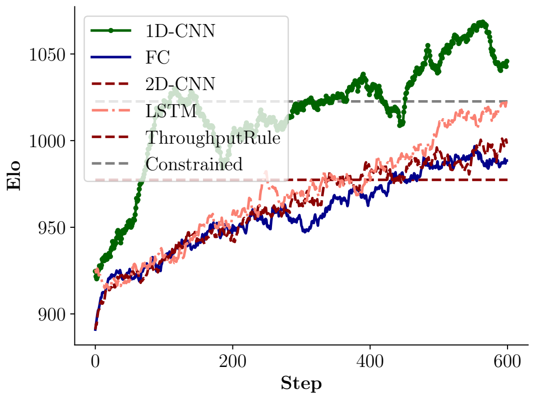

Fig. 9. Tiyuntsong's network architecture.

D.2. Tiyuntsong with Different Architectures

In this experiment, we compare the Dual network architecture from Tiyuntsong to the following network architectures which collectively represent the architecture candidates. The network architecture candidates are simply listed as follows:

- **Fully Connected**
  $\mathtt{FC}^1_{64} \rightarrow \mathtt{FC}^2_{128} \rightarrow \mathtt{FC}^3_{64}$
- **LSTM (long-short-term-memory)**
  $\mathtt{LSTM}^1_{64} \rightarrow \mathtt{LSTM}^2_{64} \rightarrow \mathtt{SELF\text{-}ATTENTION}^1_{64}$
- **2D-CNN**
  $\mathtt{CONV2D}^1_{64} \rightarrow \mathtt{MAXPOOL}^1_2 \rightarrow \mathtt{CONV2D}^2_{64} \rightarrow \mathtt{MAXPOOL}^2_2 \rightarrow \mathtt{FC}^1_{64}$
- **1D-CNN***
  $\mathtt{CONV1D}^{1\cdots 6}_{64} \rightarrow \mathtt{MERGE}^1 \rightarrow \mathtt{FC}^1_{64}$

We train and test under Sabre environment with the same network traces and video descriptions. In this experiment, we set $\gamma = 0.99$, $\beta = 0.02$, $step = 50$ for only testing their performance instead of convergence. We report the result in Figure 9 and Table 3, where 1D-CNN is the Tiyuntsong's Dual network architecture. The obtained results indicate that 1D-CNN neural network architecture succeeds in improving the Elo ratings, with improvements in average Elo ratings of 37 - 61. We also observe that there is no obvious difference between these architectures in terms of operational efficiency.

E. DISCUSSIONS

E.1. Traditional QoE functions and Rules

As demonstrated in Figure 10, we find that more than one rule can be generalized to represent the same QoE formulation, vice versa. For example, during the design of method,

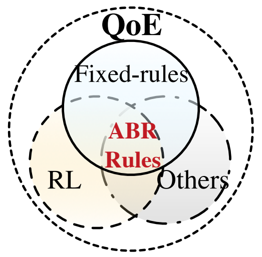

Fig. 10. Traditional RL method's Trap: A QoE metric can evaluate several ABR algorithms, but the generated algorithm may deviate from the basic rules of the ABR if it blindly improves QoE score.

fixed-rules, such as throughput-based and buffer-based, use handcraft features or network presumptions to implement the model without considering how to take advantage of evaluation metrics (QoE formulation). Then, the given QoE formulation is only used to evaluate the performance of each algorithm. Furthermore, mixed-based and RL-based schemes, i.e., MPC [4] and Pensieve [6] adopts the QoE formulation to *guide* its algorithm for achieving higher QoE score. However, recent research [**?**] exposes that there still exists a plenty of room for improving QoE metrics and many situations (e.g., some network conditions and videos) cannot be evaluated correctly via current QoE metrics due to the lack of features, as the RL-based scheme still tries to optimize the QoE score with the false guidance, finally results in failure of real-world performances. As a result, no matter how precisely and carefully the QoE function tunes, traditional RL-methods cannot exactly provide a result that the users desire.

Intuitively, the critical idea of Rule is to tackle the problem that the reward function fails to depict. For example, ABR tasks and self-driving car tasks. The fundamental factor of Rule is: Given two answers (action) from one questions (state), can *you* figure out which one is better to answer? In this paper, we prove that using self-play reinforcement learning will learn the strategy by itself if *you* can *tell* the agent who is better.

E.2. The Diversity of Network Traces

The real world network is composed of several network conditions such as 3G/HSPDA, 4G, Wired and WiFi. It's obvious that each of them has different features, and we try to train a generalized model which can cover all the network status. Thus, we collect a corpus of network traces by combining several public datasets. Meanwhile, the diversity of the length of network traces is still challenging. On the one hand, each dataset is generated in different durations and granularity. For example, each of FCC dataset we used logs the average throughput about 100 seconds, and at a 5-second granularity (the log is sized 20); Each of synthetic network traces is logged as the average throughput about 2000 seconds. On the other hand, we balance the data distribution by controlling the amount of various network traces in the data pool during the training process. Additional information will be open-sourced later on.