Learning Video Transmission Algorithms via Requirements

In Submission.

***Abstract*—Video streaming services employ video transmission algorithms to fulfill users' demands, e.g., achieving high bitrate and low rebuffering. However, existing techniques have largely used a goal, that linearly combines several weighted metrics, to generalize a strategy, of which might eventually violate the original requirement. To eliminate this concern, we propose *Zwei*, a novel self-play reinforcement learning framework for video transmission tasks. Zwei aims to update the policy towards the actual requirement, where the requirement can be described as a set of rules with priority. Technically, we apply the Monte-Carlo search algorithm to effectively sample the trajectories from the same starting point, and instantly estimate the win rate w.r.t the competition outcome. The competition result represents which trajectory is closer to the assigned requirement. Zwei uses the typical reinforcement learning algorithm to learn the policy via maximizing the win rate, as the proposed framework enables the user to leverage more apparent optimization objectives. To build Zwei, we develop video transmission simulation environments, design adequate neural network models, and invent training methods for dealing with different requirements on various video transmission scenarios. As expected, trace-driven analysis over three representative tasks (i.e., adaptive bitrate streaming, end-to-end real-time communication, and crowd-sourced live streaming scheduling) demonstrates that Zwei optimizes itself according to the assigned requirement faithfully, outperforming the state-of-the-art methods under all considered scenarios.**

***Index Terms*—Video transmission Strategy, Reinforcement Learning, Adaptive Bitrate Streaming.**

I. Introduction

Thanks to the dynamic growth of video encoding technologies and basic Internet services [1], currently humans are living with the great help of video transmission services. Users often watch interesting videos and live streaming from different video content providers (e.g. YouTube and Kuaishou), or chat with each other via real-time video communication rather than a conventional phone call. In such scenarios, the videos are required to transmit with high bitrate and less rebuffering time or stalling ratio. However, due to the fluctuation and unpredictability of network conditions, blindly achieving high bitrate may heavily increase the probabilities of the rebuffering event. Hence, several rate adaptation methods are proposed to take these factors into consideration. Further, from the video content provider's perspective, they aim to provide video streaming services with less stalling ratio but lower costs, where it's also necessary to trade off the stalling ratio against the cost. Thus, the services are required to employ scheduling algorithms for balancing the two. In brief, such aforementioned observations are being left on the horns of a classic dilemma: in the video transmission tasks, both quality of experience (QoE) and quality of service (QoS) are evaluated with contradicted metrics (§II-A). Hence, how to generally bridge the gap between the positive and negative factors?

Unfortunately, as much as the fundamental problem has already been published about *two decades* [2], [3], current approaches, either heuristics or learning-based methods, fall short of achieving this goal. On the one hand, heuristic-based schemes often use existing models [4], [5] or specific domain knowledge [6] as the basic working principle. However, such approaches sometimes require careful tuning and will backfire under the circumstance that violated with presumptions, which results in the failure of achieving acceptable performance under all considered scenarios [7]. On the other hand, learning-based methods [8], [9] leverage deep reinforcement learning (DRL) to train a neural network (NN) by interacting with the environments without any presumptions, aiming to obtain higher score computed by the reward function, where the function is often defined as a linear-based equation with the combination of weighted sum metrics. Unsurprisingly, the learned policy outperforms heuristics, with the improvements on the overall performance of more than 18% [10]. Nevertheless, in this study, we empirically illustrate that i) an inaccurate reward function may mislead the RL-based algorithm to generalize bad strategies, since ii) the actual requirement can hardly be presented by the linear-based reward function with fixed weights. Moreover, iii) considering the diversity of real-world network environments, we can hardly present an accurate reward function that can perfectly fit *any* network conditions (§II-B). As a result, despite its abilities to gain higher numerical reward score, such training schemes may generalize a strategy that hardly meet the basic rules of the actual requirements.

Taking a look from another perspective, we observe that the aforementioned problem can be naturally written as a deterministic goal or requirement [11]. E.g., in the most cases, the goal of the adaptive bitrate (ABR) streaming algorithm is to achieve lower rebuffering time first, and next, reaching higher bitrate [12], [13]. It is pretty straightforward that it can be easily understood and refined by others. To this end, we attempt to train the NN based on the assigned requirement without reward engineering. Unfortunately, off-the-shelf learning-based algorithms are unable to optimize the policy like this since it lacks the abilities to provide any gradients information to guide the algorithm towards a better performance directly. Hence, we ask if self-play learning [14] can help taming the complexity of video transmission services with the actual requirement and especially, without reward engineering.

Corresponding Author: xxx, mailto: xxx.
xxx, xxx.
Manuscript received July 21, 2019; revised Oct. 26, 2019.

Inspired by this opportunity, we envision a self-play reinforcement learning-based framework, known as Zwei [1], which can be viewed as a solution for tackling the video transmission dilemma (§III). The key idea of Zwei is to generalize a strategy which can always meet the actual requirement. Specifically, we apply Monte-Carlo (MC) search to making move decisions in the game of backgammon. In the MC search process, many simulated trajectories starting from the starting state s_0 are generated following current policy θ, and the expected long-term win rate r is estimated by averaging the battle results from each of the trajectories. The battle result are determined by a basic question: given two strategies collected from the same environment, which one is closer to the actual demand? Having estimated the long-term win rate, Zwei then updates the NN via increasing the probabilities of the winning sample and reducing the possibilities of the failure sample. In this work, we use state-of-the-art policy gradient method proximate policy optimization (PPO) [15] to optimize the NN. Furthermore, inspired by the recent success of state-of-the-art RL methods [16], [15], we further add NN-based baseline into Zwei to enhance its overall performance (§III-B).

In the rest of the paper (§IV), we attempt to evaluate the potential of Zwei using trace-driven analyses of various representative video transmission scenarios. To achieve this, we build several faithful video transmission simulators which can accurately replicate the environment via real-world network dataset. Specifically, we validate Zwei on three different tasks (§II-A), including client-to-server, server-to-client, and client-to-client service. Note that each of them has individual requirements and challenges. As expected, evaluation results demonstrate the superiority of Zwei against existing state-of-the-art approaches on all tasks. In detail, we show that **(i)** Zwei outperforms existing ABR algorithms on both HD videos and 4K videos, with the improvements on Elo score [17] of 32.24% - 36.38%. **(ii)** Zwei performs well in crowd-sourced live streaming (CLS) scheduling task, reducing the overall costs on 22% and decreasing over 6.5% on the overall stalling ratio in comparison of state-of-the-art learning-based scheduling method LTS [18]. **(iii)** Zwei generalizes well in the real-time communication (RTC) scenario. Comparing the performance of Zwei and state-of-the-art heuristics WebRTC [19], we observe that Zwei effectively improves 11.34% on average receive rate, but reduces 20.49% on loss ratio and 62.95% on 95-percent round-trip-time (RTT). In particular, we find that Zwei perfectly follows the guidance of the given requirement since it demonstrates two distinct stages during the training process.

Contributions: This paper makes four key contributions.

- We point out the shortcoming of learning-based schemes in video transmission tasks and present the idea that update networks without reward engineering (§II-B).
- Zwei is a novel framework that aims to employ the self-play reinforcement learning method to make the idea practical (§III).
- We implement Zwei into three representative video transmission scenarios, including rate adaptation, transmit scheduling, as well as rate control. For each task, we have to develop new NN representation, determine requirements, and construct a faithful offline simulator (§IV).
- We evaluate Zwei under video transmission tasks. Trace-driven experimental results illustrate that Zwei outperforms existing schemes on all considered scenarios (§IV).

[1]Zwei: means the number *two*, or *double* in German

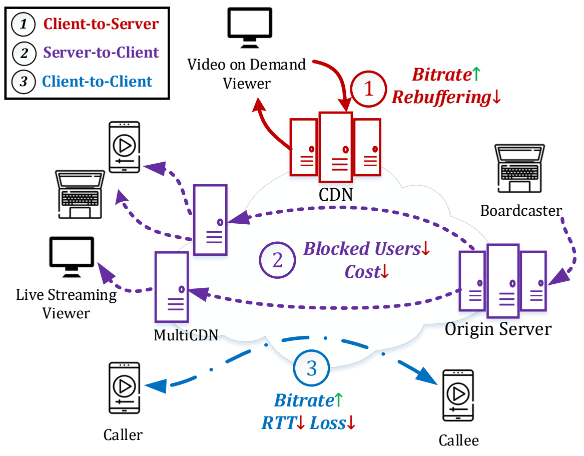


Fig. 1. A brief introduction of today's video transmission service. As shown, the service is mainly composed of client-to-server, server-to-client and client-to-client services (§II-A).

II. Background and Challenges

In this section, we first normally introduce the background of adaptive video streaming scenarios, including client-to-server, server-to-client, and client-to-client service, and then we briefly elaborate the key challenges of each service.

A. Video Transmission Services

To better understand the limitations that conventional video streaming service suffering from, we plot the general service work-flow on Figure 1. We summarize the task name, task features, requirement and constrains for each video transmission scenario in Table I. As shown, the service commonly consists of three parts:

Client-to-Server Service. In this scenario, users often adopts a video player to watch the video on demand. First, video contents are pre-encoded and pre-chunked as several bitrate ladders on the server. Then the video player, placed on the client side, dynamically picks the proper bitrate for the next chunk to varying network conditions. Specifically, the bitrate decisions should achieve high bitrate and low rebuffering on the entire session [20], [12]. We called it *adaptive bitrate streaming* (ABR).

Server-to-Client Service. Consider, if we were the content provider and currently we had multiple content delivery networks (CDNs) with different costs and performance, how to schedule the users' requests to the proper CDN, aiming to provide live streaming services withing less stalling ratio and lower cost? In common, we call that *crowd-sourced live streaming(CLS)* [18].

Client-to-Client Service. Besides, in our daily life, we usually chat with other users instantly via a video call namely

TABLE I
REQUIREMENTS FOR EACH VIDEO TRANSMISSION TASKS.

Transmission Type	Is ABR?	Requirement
Client-to-Server	✓	Rebuffering Time[1]↓, Bitrate2↑
Server-to-Client	×	Stalling Ratio[1]↓, Cost[2]↓
Client-to-Client	✓	RTT[1]↓, Loss[1]↓, Bitrate[2]↑

Real-Time video Communication (RTC). The RTC system consists of a sender and a receiver. The sender deploys a UDP socket channel and sends the encoded video packets to the receiver. The receiver then feeds the messages back to the sender through feedback channel. During the session, the sender adjusts the sending bitrate for next time period, aiming to achieve high bitrate and low round-trip time (RTT) as well as less loss ratio [9].

In general, we list the requirement and details for each video transmission task in Table I. Notice that we label the underlying metric orderly since each metric has attentive priority (E.g, the priority of low rebuffering time is higher than that of high bitrate in the ABR scenario). Details of each video transmission scenario are demonstrated in §IV. Obviously, the video transmission task algorithm is often required to obtain better performance under various mutual metrics, which makes the key principle of designing a traditional networking algorithm: best-effort delivery [3]. While learning-based approaches pursue best-performance delivery, aiming to maximize the overall performance over all considered networks. To this end, what's the Achilles' heel of the best-performance approach?

B. Challenges

In this work, we aim to fuse the best-effort approach (i.e., heuristics) and best-performance approach (i.e., learning-based approach). Recent video transmission algorithms mainly consist of two types, i.e., heuristics and learning-based scheme. Heuristic methods utilize an existing fixed model or domain knowledge to construct the algorithm, while they inevitably fail to achieve optimal performance in all considered network conditions due to the inefficient parameter tuning (§V-A,[8]). Learning-based schemes train a NN model towards the higher reward score from scratch, where the reward function is often defined as a linear combination of several weighted metrics [18], [9], [8]. Nevertheless, considering the characteristics of the video transmission tasks above, *we argue that the policy generated by the linear-based reward fails to always perform on the right track [11].* In this study, we set up two experiments in the ABR scenario (IV-A) to prove this conjecture. The reason why we use this scenario to describe the challenges is that the ABR task is the easiest to understand and closest to the user in the video transmission scenario [1].

Observation1. *The best learning-based ABR algorithm Pensieve [8] is not always the best scheme on every network traces.*

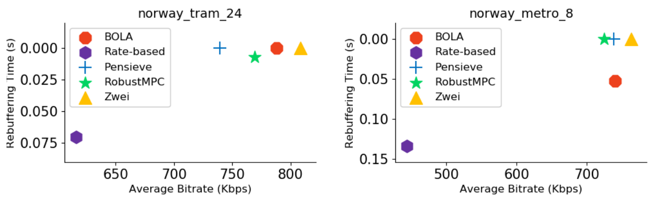


Fig. 2. We show average bitrate and rebuffering time for each ABR method. ABRs are performed over the HSDPA network traces.

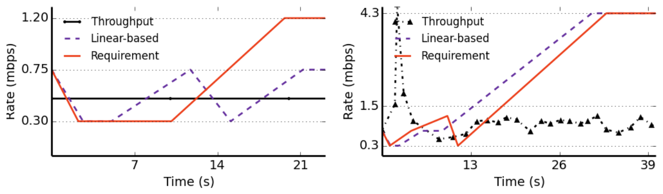


Fig. 3. The comparison of linear-based optimal and requirement-based optimal strategy. Results are evaluated on *fixed (0.5mbps)* and *HSDPA* [24] network traces under ABR scenario, and we can see the difference between the actual requirement and the optimal trajectory generated by the linear-based reward function.

Given a deterministic QoE metric with linearly combining of several underlying metrics [21], [22], in recent years, many schemes considered the ABR process as a Markov Decision Process (MDP) and have been proposed to learn ABR algorithms via reinforcement learning (RL) method. E.g., RobustMPC [21] is a model-based RL approach which utilizes a solver (e.g. `CPLEX` or `Gurobi`) and a faithful offline ABR simulator to maximize QoE objectives by planning; Pensieve [8] is a model-free RL approach which learns the system dynamic via interacting with the ABR environment. However, such method heavily relies on the accuracy of the given QoE metric. Especially, *how to set a proper QoE parameter for each network condition* is indeed a critical challenges for training a good ABR algorithm. In order to verify whether QoE parameters have influenced the performance of ABR algorithms, we set up an experiment to report average bitrate and rebuffering time for each ABR method, including Rate-based, BOLA, Pensieve, RobustMPC, and Zwei ([23], [21], [8], §IV-A3). Results are evaluated over the HSDPA network traces. We can see that, despite the outstanding performances that Pensieve achieve, the best RL-based ABR algorithm does not always stand for the best scheme. Note that Zwei always outperforms existing approaches.

Observation2. *Recent linear-based weighted reward function can hardly map the actual requirement for all the network traces.*

To better understand the effectiveness of weighted-sum-based reward functions, we compare the optimal strategy of linear-based with requirement-based under two representative network traces, in which linear-based optimal is the policy which obtains maximum reward, and the requirement-based optimal stands for the closest strategy in terms of the given requirement. Unsurprisingly, from Figure 3 we observe that

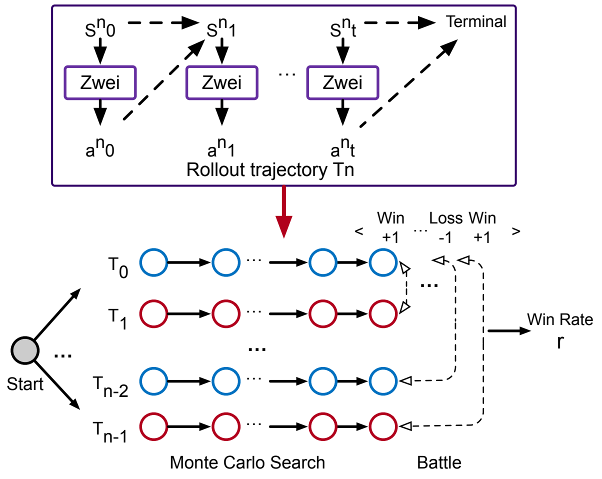


Fig. 4. Overview of Self-play Reinforcement Learning Framework (Zwei). The framework is mainly composed of *four* phases: Monte Carlo (MC) searching, battle competition, win rate estimation, and policy optimization.

linear-based optimal policy performs differently compared with the requirement-based optimal strategy. The reason is that the linear-based optimal policy heavily relies on the given weights, while the weighted parameters are not allowed to be adjusted dynamically according to current network status (e.g., throughput and jitter [10]), which finally leads to the failure of guiding the optimization process on the right track. Generally, we believe that the policy learned by reward engineering might fall into the unexpected conditions.

Summary. We observe that no matter how precisely and carefully the parameter of the linear-based reward function tunes, such function can hardly meet the requirement of any network conditions. E.g., the parameter of stable and unstable network conditions are not the same. To that end, traditional learning-based scheme, which often optimizes the NN via the assigned functions, will eventually fail to provide a reliable performance on any network traces.

In this paper, we attempt to generalize the strategy based on actual requirements instead of linear-based reward functions. Considering the basic requirement cannot provide any gradients to the NN, we, therefore, treat the learning task as a competition between two trajectories sampled from the same policy and environment settings, and ask if self-play learning can help taming video transmission problems based on actual requirements, and especially, without using reward engineering.

III. Zwei Design

In this section, we briefly introduce the details of Zwei, including its key principle,the basic training algorithm and the advanced improvements.

A. Self-play Method

AlphaZero [25] is the most famous self-play algorithm who trains the NN solely based on an agent competes against the others. To tackle the traditional RL methods' problem, we thereby consider following AlphaZero to train the algorithm via self-play RL. The main idea of our proposed framework is to leverage these search operators repeatedly in a policy iteration procedure. However, static games with incomplete information (i.e., video transmission tasks are often defined as partially observable Markov decision process (POMDP) problems [7].) are much dissimilar and more complicated than dynamic games with complete information such as Go or Chess. Thus, we propose a novel self-play reinforcement learning framework, namely Zwei, aiming to tackle a circtical challenge: *considering the specific features for video transmission task, how to design a proper general self-play reinforcement learning algorithm?*

B. Basic Idea

We now start to explaining the fundamental idea for Zwei. Figure 4 presents the main phases in our framework. The pipeline can be summarized as follows:

Phase1: Monte Carlo (MC) search. First, we adopt MC search method [26] to sample N different trajectories $T_n = \{s^n_0, a^n_0, s^n_1, a^n_1, \dots, a^n_t\}, n \in N$ w.r.t the given policy $\pi(s)$ under the *same environments* ($a_t \sim \pi(s_t)$) and start point (the gray point in Figure 4). Next, we record and analysis the underlying metric for the entire session. Finally, we store all the sample T_n into D. Note that we can select Monte Carlo Tree Search (MCTS) [14], which is widely used in advanced research, to implement the process.

Phase2: Battle Competition (BC). To better estimate how the current policy performs, Zwei requires a module to label all trajectories from D: given two *different* trajectories, T_i and T_j which are all collected from the *same environment settings* ($T_i, Tj \in D$)), we attempt to identify which trajectory is positive for NN updating, and which trajectory is generated by the worse policy. Thus, we implement a rule called `Rule` which can determine the *better* trajectory between the given two candidates, in which *better* means which trajectory is closer to the requirement. At the end of the session, the terminal position s_t is scored w.r.t the rules of the requirement for computing the game outcome o: -1 for a loss, 0 for a draw, and $+1$ for a win. The equation is listed in Eq. 1.

$$o^j_i = \texttt{Rule}(T_i, T_j). \tag{1}$$

$$s.t. \quad o^j_i = \{-1, 0, 1\}, T_i, T_j \in D, \tag{2}$$

$$T_i \neq T_j. \tag{3}$$

Phase3: Win Rate Estimation. Next, having computed the competition outcome o_i for any two trajectories, we then attempt to estimate the average win rate r_i for each trajectory T_i in D. The equation is listed in Eq. 4. Notice that the accuracy of the win rate estimation heavily depends on the number N of trajectories. Since it's impractical to sample infinite number of samples in the real world, we further list the performance comparison of different sample numbers in §IV-A.

$$r_i = \mathbb{E}[\texttt{Rule}(T_i,)] \tag{4}$$

$$= \lim_{N \to \infty} \frac{\sum_u^N o^u_i}{N}. \tag{5}$$

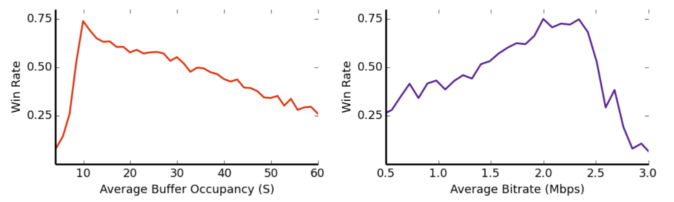


Fig. 5. We plot the underlying win rate for different average bitrate and average buffer occupancy.

Phase4: Policy Optimization. In this part, given a batch of collected samples and their win rate, our key idea is to update the policy via *elevating* the probabilities of the winning sample from the collected trajectories and *diminishing* the possibilities of the failure sample from the worse trajectories. In other words, the improved policy π at state s_t is required to pick the action a_t which produced the best estimated win rate r_t, i.e., $a_t = argmax_a E[r_t(s_t, a)]$. Hence, we can employ any policy gradient method (e.g., A3C [16], TRPO [27], ACKTR [28]) to optimize the NN's policy. In this work, We use Proxy Policy Optimization (PPO) [15], a state-of-the-art actor-critic method, as the basic RL algorithm. Briefly, PPO uses clip method to restrict the step size of the policy iteration and update the NN by minimizing the following *clipped surrogate objective*. We list the Zwei's loss function $\mathcal{L}^{\text{PPO}}(\theta)$ in Eq. (6).

$$\mathcal{L}^{\text{PPO}}(\theta) = \hat{\mathbb{E}}_t\Big[\min\Big(p_t(\theta)r_t, \text{clip}\big(p_t(\theta), 1-\epsilon, 1+\epsilon\big)r_t\Big)\Big], \tag{6}$$

where $p_t(\theta)$ denote the probability ratio between the policy π_θ and the old policy $\pi_{\theta_{\text{old}}}$, i.e., $\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$; ϵ is the hyper-parameters which controls the clip range. In this paper, we set $\epsilon = 0.2$ as suggested by the original PPO paper [15].

As listed before, Zwei's basic training algorithm is derived from a fundamental presumption: the win rate of each state s_t will be converged to zero, that means, *a draw game*. However, we argue that the distribution of win rate on each state s is different. Thus, we setup an experiment in the ABR scenario: we select a short video clip encoded with 6 bitrate levels and 5 chunks, and list all the possible trajectory under the same network trace. Then we compute the underlying win rate for each state s_t via `Rule`, where s_t is represented by *Average Bitrate* and *Average Buffer*. This makes sense since recent work have proved that these two metrics are critical feature for ABR task [22]. Experimental results on Figure 5 illustrate that *different states map different win rates*. E.g., the win rate will degrade to 25% if average bitrate is lower than 1 mbps and the average buffer is higher than 40 seconds. At the same time, win rate will achieve the highest score under the conditions that average bitrate is about 2 mbps and the average buffer is in the range of 10 seconds to 20 seconds. What's more, we also consider the correlation between the average buffer and win rate, as well as the relationship between average bitrate and win rate respectively. Results also indicate the influence on win rate by different values of current states.

To this end, we further add the additional baseline $V_{\theta_p}(s)$ to avoid the bias which is caused by the high variance in each situation. In details, Zwei with Baselines consists of a policy network and a value network. The gradient of policy network are computed as Eq. 7, where $\hat{A}_t$ is the advantage function (Eq. 8), $V_{\theta_p}(s)$ is provided by another value NN. Meanwhile, we update the value network via minimizing the mean square error between r_t and $V_{\theta_p}(s)$ (Eq. 9). Moreover, we also add $H(s_t;\theta)$, representing the entropy of the policy, to encourage exploration. In this work, we use adaptive entropy decay method to dynamically adjust the entropy weight β from 0.5 to 0.01. For more details, please refer to our repository [29]. To sum up, we summarize Zwei's loss function $\mathcal{L}^{Zwei}$ in Eq. 10.

$$\mathcal{L}^{\textit{Policy}} = \hat{\mathbb{E}}_t\Big[\min\Big(p_t(\theta)\hat{A}_t, \text{clip}\big(p_t(\theta), 1-\epsilon, 1+\epsilon\big)\hat{A}_t\Big)\Big]. \tag{7}$$

$$\hat{A}_t = r_t - V_{\theta_p}(s_t) \tag{8}$$

$$\mathcal{L}^{\textit{Value}} = \frac{1}{2}\hat{\mathbb{E}}_t\left[V_{\theta_p}(s_t) - r_t\right]^2. \tag{9}$$

$$\nabla\mathcal{L}^{Zwei} = \nabla_\theta\mathcal{L}^{\textit{Policy}} + \nabla_{\theta_p}\mathcal{L}^{\textit{Value}} + \nabla_\theta\beta H(s_t;\theta). \tag{10}$$

C. *Training Methodology Overview*

Training procedure of Zwei is summarized in Algorithm 2. As shown, Zwei's workflow mainly consists of two parts, i.e., exploration and exploitation:

1) *Exploration.* First, we sample n trajectories roll-outed by the current policy in the same environment. Then we leverage `Rule` to determine the better strategy via the pairwise comparison. Next, we estimate the win rate w.r.t the given samples.
2) *Exploitation.* We update the gradient by using the loss function $\mathcal{L}^{Zwei}$ (Eq. 10).

D. *Parallel Training*

During the training process, we observe that the training progress is inefficient while using a single process. Inspired by the multi-agent training method [16], we modify Zwei'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 $\{s, a, r\}$ 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, Zwei employs 12 forward propagation agents and one central agent.

Algorithm 1 Zwei's Overall Training Procedure

```
Require: Environment Env; Rule Rule.
1:  Initialize parameters $\theta$ with random weights;
2:  repeat
3:      $D = \{\}$;
4:      Env ←Sample environment settings, e.g., network trace T, video V, or CDN configuration $c$;
5:      for $i \in \{1, 2, \ldots, N\}$ do                         ▷ MC search;
6:          Get state $s_t$ from Env.
7:          $T_i = \{\}$;
8:          while not Terminal do
9:              Perform $a_t$ according to policy $\pi_\theta(a_t; s_t)$.
10:             $T_i \leftarrow T_i \bigcup \{s_t, a_t\}$.
11:             Receive new state $s_{t+1}$
12:         $D \leftarrow D \bigcup T_i$
13:     for $pairs(T_u, T_v) \in D$ do                  ▷ Battle competition;
14:         Determine win or loss: $o_u^v = \mathtt{Rule}(T_u, T_v)$;
15:     for $T_u \in D$ do                                ▷ Win rate estimation;
16:         Estimate win rate: $r_u = \frac{\sum_i^N o_u^i}{N}$.
17:         for $s_t, a_t \in T_u$ do                     ▷ Policy Optimization;
18:             Update $\theta$ and $\theta_p$ with $\mathcal{L}^{Zwei}$ (Eq. 10) using collected samples and win rate $(s_t, a_t, r_u)$;
19: until Converged
```

E. Implementation

We use TFlearn [30] to implement the NN and leverage TensorFlow [31] to construct Zwei. Zwei consists of two NNs, policy network and value network. Policy network takes a n-dims vector with *softmax* active function as the output. The value network outputs a value with *tanh* function scaled in $(-1, 1)$. Considering the characteristics on video transmission tasks, we construct different NN architectures for each task. In contrast, we use the same set of hyper-parameters for training the NN: sample number $N = 16$, learning rate $\alpha = 10^{-4}$, and the entropy weight decay $\zeta = 0.8$. In addition, we adopt Adam optimizer [32] with default settings. Notice that we will discuss the NN architecture and the effect of different sample number in §IV-A.

IV. Evaluation

In this section, we thoroughly evaluate the performance of the schemes which utilizes Zwei to train the NN. We mainly consider three tasks (i.e., ABR , RTC, and CLS tasks) which have been described before (§II-A).

A. ABR Scenarios

We first understand how Zwei works in the traditional ABR scenario. Detailing the working process, that includes, testbed setup, baseline introduction, and Zwei vs. Existing ABR schemes.

1) ABR's Background: 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) [33] and DASH [34], an algorithm that dynamically selects video bitrates via network

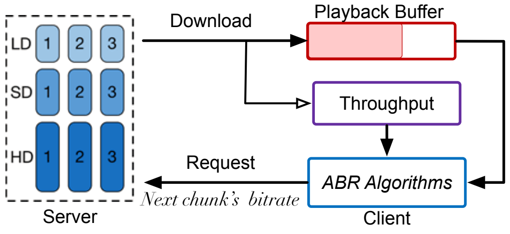


Fig. 6. This figure shows the typical ABR System.

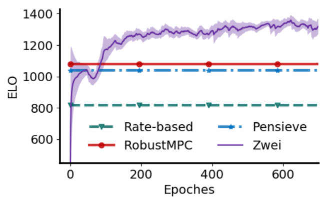
conditions and client's buffer occupancy, is the predominant form of video delivery [20]. The traditional video streaming architecture is shown in Figure 6. It consists of a video player client with a constrained buffer length and an HTTP-Server or Content Delivery Network (CDN). The video player client 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.

2) Zwei's NN Architecture for ABR Tasks: We now explain the details of the Zwei's neural network (NN), including its inputs, outputs, network architecture, and implementation.

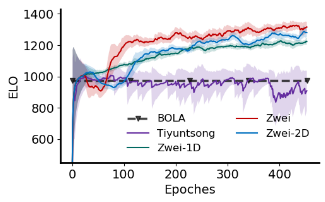
Inputs. The NN's inputs is mainly categorized into three parts, network features, video content features and video playback features ($S_k = \{C_k, M_k, F_k\}$). We takes past chunk $k = 8$ as the input. NN takes past t chunks' network status vector $C_k = \{c_{k-t-1}, \ldots, c_k\}$ into NN, where c_i represents the throughput measured for video chunk i. Specifically, c_i is computed by $c_i = n_{r,i}/d_i$, in which $n_{r,i}$ is the downloaded video size of chunk i with selected bitrates r, and d_i means download time for video chunk $n_{r,i}$. Besides, we also consider adding video content features into NN's inputs for improving its abilities on detecting the diversity of video contents. In details, the learning agent leverages $M_k = \{N_{k+1}\}$ to represent video content features. Here N_{k+1} is a vector that reflects the video size for each bitrate of the next chunk $k+1$. The last essential feature for describing the ABR's state is the current video playback status. The status is represented as $F_k = \{v_{k-1}, B_k, D_k, m_k\}$, where v_{k-1} is the perceptual video quality metric for the past video chunk selected, B_k, D_k are vectors which stand for past t chunks' buffer occupancy and download time, and m_k means the normalized video chunk remaining.

Outputs. We attempt to use discrete action space to describe the output. Note that the output is an n-dim vector indicating the probability of the bitrate being selected under the current ABR state S_k. In this work, we set $n = 6$, which is widely used in ABR papers [11], [35], [8], [10], [36].

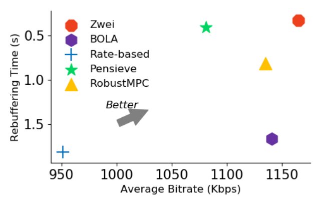
NN Representation. Zwei's NN representation is quite simple: it leverages three fully-connected layers, which is sized 128, 64, and 64 respectively, for describing the feature



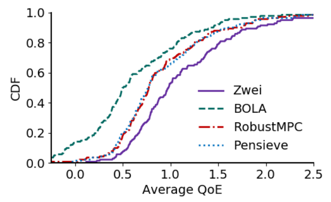
(a) Zwei vs. ABR schemes



(b) Zwei vs. Tiyuntsong



(c) ABR Scheme Details



(d) CDF of ABR Algorithms

Fig. 7. This group of figures show the comparison results of Zwei and other ABR approaches. Results are evaluated in the typical ABR system with HD Videos (video resolution=1920 $\times$ 1080, maximum bitrate=4.3mbps).

extraction layer. On the one side, the output of the NN's policy network is a *6-dims* vector, which represents the probabilities for each bitrate selected. We utilize *RelU* as the active function for each feature extraction layer and leverage *softmax* for the last layer. On the other size, NN's value network outputs a scalar, which uses *tanh* as the active function.

Requirements for ABR tasks. Algorithm 2 is an example of Rule `Rule` which used in the ABR scenario, where ϵ_c is a small number that can add noise for improving Zwei's robustness. In this work, we set ϵ_c as 10. Note that, such settings is a non-trivial metric for Zwei.

Algorithm 2 Rule for the ABR task.

```
Require: Trajectory $T_u, T_v$;
1: Compute underlying metric average bitrate $r_u, r_v$, average rebuffering $b_u, b_v$ from the given trajectories $T_u, T_v$.    ▷ Requirements: i) low rebuffering time ii) high bitrate.
2: Initialize Return $s = \{-1, -1\}$;
3: if $|b_u - b_v| < \epsilon_c$ or $|r_u - r_v| < \epsilon_c$ then
4:     Randomly set $s_0$ or $s_1$ as 1.    ▷ Add noise to improve the robustness.
5: else if $|b_u - b_v| < \epsilon_c$ then
6:     $s_i \leftarrow 1, i = argmax_{i \in \{u,v\}} r$;
7: else
8:     $s_i \leftarrow 1, i = argmin_{i \in \{u,v\}} b$;
9: return $s$;
```

QoE Representation. Recall that the goal of ABR algorithm is to select bitrates for the next chunk k with high bitrate r_k and less rebuffering time t_k [13], thus the optimization reward q_k can be normally written as Eq. 11.

$$q_k = r_k - \alpha t_k, \tag{11}$$

here α is the coefficient that adjust the importance of the underlying metrics. In practice, α is often defined as 3 [21], 4.3 [8], or 6.0 [21]. Prior research add additional smoothness metric $|r_k - r_{k-1}|$ to control the bitrate change of the entire session, while in practice the following metric is neglectable for the ABR algorithm [13], [12]. Hence, in this work we also set the smoothness to zero for better understanding the fundamental performance of Zwei. We believe that Zwei can also solve q_k with smoothness metric, e.g., the action space can be set as relative bitrate moves.

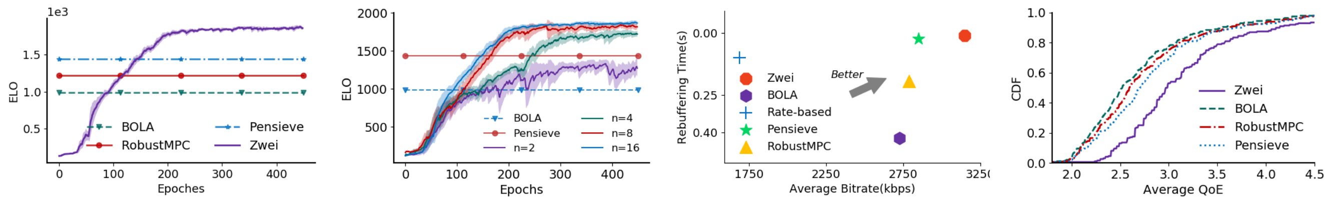
3) ABR Baselines: In this paper, we select several representational ABR algorithms from various type of fundamental principles:

1) ***Rate-based* [37]:** the basic baseline for ABR problems. It leverages *harmonic mean* of past five throughput measured as future bandwidth.
2) ***BOLA* [23]:** the most popular buffer-based ABR scheme in practice. BOLA turns the ABR problem into a utility maximization problem and solve it by using the Lyapunov function. We use BOLA provided by the authors [38].
3) ***RobustMPC* [21]:** a state-of-the-art heuristic method which maximizes the objectives by jointly considered the buffer occupancy and throughput predictions. We implement *RobustMPC* by ourselves.
4) ***Pensieve* [8]:** the state-of-the-art learning-based ABR scheme, which utilizes DRL to select bitrate for next video chunks. We use the pre-trained Pensieve model provided by the authors [39].
5) ***Tiyuntsong* [11]:** the first study of multi-objective optimization ABR approach. Tiyuntsong uses actor-critic method to update the NN via the competition with two agents under the same network condition. Note that Tiyuntsong's learning agent uses DRL to update gradients w.r.t the sample generated by the agent itself, which not only sample inefficient but also reaching the optimal policy.

4) Experimental Setup: In this part, we utilize the standard ABR's emulation environment, including Mahimahi [40] and Park [41], to evaluate Zwei. In details, we train and validate Zwei on various network throughput dataset, such as HSDPA [24], FCC [42] and Oboe [10]. We use the video dataset provided by Huang et al. [35] during the training process. Training process lasted approximate 45000 steps, within 34 hours to obtain a reliable result. In this experiment, we setup two ABR scenarios, i.e., HD and 4K video scenario. In the HD video scenario, we adopt *EnvivoDash3*, a video that commonly used in recent work [10], [8], [35], to validate Zwei, where the video chunks are encoded as $\{0.3, 0.75, 1.2, 1.8, 2.8, 4.3\}$ mbps. In the 4K-video scenario, we use the popular open-source movie Big Buck Bunny [43], which is now even more a world standard for video standards. Specifically, we pick 6 bitrates from `Dash.js` standard [34], i.e., $\{0.2, 0.6, 1.5, 4.0, 8.0, 12.0\}$ mbps. [2]

5) Evaluation Metrics: The Elo rating [45] is a traditional method for calculating the relative performance of players in zero-sum games. It's suitable to compare different schemes via win rate information only. Thus, we also use Elo score to com-

[2]We deleted some unnecessary bitrates that RL-based scheme seldom picks, e.g., $\{1.2, 2.85\}$ mbps [44].



(a) Zwei vs. ABR schemes (b) Zwei with Different Samples (c) ABR Algorithm Details (d) CDF of ABR Algorithms

Fig. 8. This group of figures show the comparison results of Zwei and other ABR approaches. Results are evaluated in the typical ABR system with 4K Videos (video resolution=3840 $\times$ 2160, maximum bitrate=12.0mbps).

pare different ABR schemes, similar to that of AlphaGo [46] and Tiyuntsong [11]. We set initial Elo rating as 1000.

6) HD Video Scenarios: First, we study the performance of Zwei over the HD video dataset and HSDPA network traces, where the video chunk are encoded as $\{0.3, 0.75, 1.2, 1.8, 2.8, 4.3\}$ mbps.

QoE metrics for other ABR algorithms. Followed by recent work [21], [36], we set $\alpha = 4.3$ as the basic QoE-HD baselines.

Pensieve Re-training. We employ Pensieve-PPO [47] to retrain Pensieve with the Pensieve training dataset, envivo3 video description, and QoE-HD metrics. Consistent with prior work [10], [35], we set different entropy weights in the range of 5.0 to 0.1 and dynamically decrease the weight with entropy decay= 0.6. Hence, it will take less than 3 hours to obtain a reliable result.

Zwei vs. Existing schemes. In this experiment, we compare Zwei with existing ABR schemes over the HSDPA dataset. Results are computed as Elo-score [17] and reported in Figure 7(a). Specifically, we first select several previously proposed approaches and validate the performance respectively under the *same* network and video environment. Next, we use `Rule` to estimate their winning rate. Finally, we compute the Elo rating for these approaches.

Through the experiments we can see that Zwei outperforms recent ABR approaches. In particular, Zwei improves Elo-score on 36.38% compared with state-of-the-art learning-based method Pensieve, and increases 31.11% in terms of state-of-the-art heuristic method RobustMPC. Besides, we also illustrate the CDF of the proposed methods on average bitrate and average rebuffering in Figure 7(c). Results show that Zwei can not only achieve highest bitrate but also obtain lowest rebuffering under all network traces. Moreover, we also compare Zwei with previously proposed self-learning method Tiyuntsong on the same experimental settings. During the training process, we report the Elo-curve on the same validation set in Figure 7(b). As expected, Zwei with any NN architectures outperform Tiyuntsong on average Elo-score of 35%. It's worth noting that, Tiyuntsong is also a self-play learning method which uses conventional actor-critic method to update the NN for obtaining a higher Q_t, where Q_t is cumulative reward $Q_t = r_t + \gamma Q_{t+1}$, where $\gamma = 0.99$. Tiyuntsong is the special case of Zwei. Thus, though Zwei's training algorithm is equal to original actor-critic methods with $r_t = 0$ and n-step discounted factor $\gamma = 1$ in mathematics, their fundamental principle are absolutely different.

Zwei with Different NN Architectures. This experiment evaluates Zwei with all considered NN architectures and compares their performance under the same network setting. Zwei-1D is the standard ABR NN architecture which proposed by Pensieve [8], Zwei-2D stands for the advanced ABR NN architecture that proposed by QARC [9], and Zwei only leverages three fully-connected layers sized $\{128, 64, 64\}$. Experimental results are calculated with Elo-score and shown in Figure 7(b). Unsurprisingly, when Zwei trains with some complicated NN architecture (Zwei-1D and Zwei-2D), it generalizes poorly and performs worse than the fully connected NN scheme. This makes sense since ABR is indeed a light-weighted task which can be solved in a practical and uncomplicated manner instead of a NN incorporating some *deep* yet *wide* layers [44].

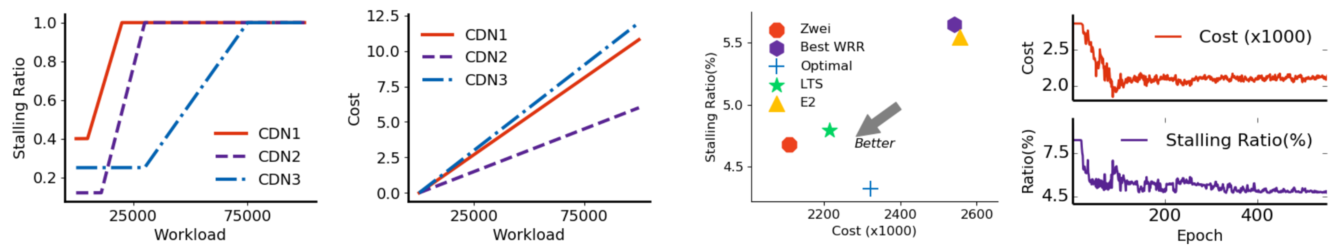
The Two-Stage Phenomenon. By meticulously observing Zwei's training curve on Figure 7(a), we find the training process can be split into two stages: 1) Zwei picked a lower bitrate to avoid rebuffering time when the training step is less than 100. Then 2) Zwei realized a new strategy and increased the bitrate with guaranteeing rebuffering events. The reason is Zwei skillfully exploits catastrophic forgetting [48] and abandons far-reaching decisions.

7) 4K Video Evaluation: Next, the second ABR scenario, we evaluate Zwei in 4K videos over HDFS [38] network traces.

QoE metrics for other approaches. Due to the difference between the maximum bitrate of 4K video (i.e., 12mbps) and HD video (i.e., 4.3mbps), we reset the QoE parameters (Eq. 11) for QoE-based ABR algorithms. In order to better avoid the occurrence of rebuffering, we set the penalty of rebuffering parameter α to 20.

Pensieve Re-training. We also use Pensieve-PPO [47] to retrain Pensieve with the HDFS [38] network dataset, the Big Buck Bunny video description, and QoE-4K. Results illustrate that Pensieve outperforms RobustMPC, with an overall average QoE improvements of 2.67% across all sessions. Meanwhile, Pensieve also performs better than RobustMPC on Elo score of 17.66%.

Zwei vs. Recent ABR Schemes. Figure 8(a) plots the learning curves in terms of the comparison of Zwei and other ABR methods. We observe that Zwei has already achieved state-of-the-art performance in almost 200 epochs, and finally, the performance improvement of Zwei can achieve 32.24% against the second best algorithm Pensieve and 58.58%-120.76% against others. What's more, as depicted in Figure 8(c), no matter average bitrate or total rebuffering time, Zwei always



(a) Piece-wise Linear Model (b) Workload - Cost (c) Zwei vs. Existing Methods (d) Zwei Training Curve

Fig. 9. Results of Zwei on the CLS environment. We collect Zwei's training curve during the training process and observe that there also exists *two stages* on the entire process.

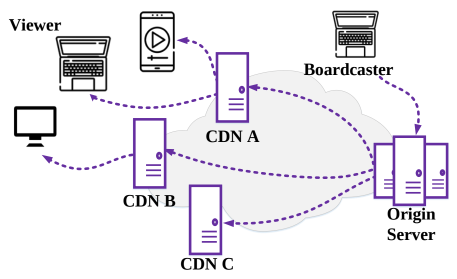


Fig. 10. CLS System overview.

stands for the best scheme among all candidates. In particular, Zwei increases 10.46% on average bitrates with reducing 56.52% on total rebuffering time. Meanwhile, we also report the comparison of QoE performance of the proposed scheme in Figure 8(d). As expected, Zwei outperforms existing ABR algorithms without reward engineering, with the increasing on average QoE of 11.59%.

Zwei with Different Samples N. Besides, we also investigate the Elo-rating of Zwei with different sample N, where we set $N = \{2, 4, 8, 16\}$. Figure 8(b) illustrates the comparison of each method, we can see that with the increase of N, Zwei can achieve *sample efficiency* [49] through variance reduction [50].

B. CLS Scheduling

In this part, we evaluate Zwei in CLS scheduling task and compare it with several state-of-the-art scheduling algorithms. Results illustrate that Zwei is able to leverage historical CDN information for providing the live video streaming service with less number of blocked (or stalling) users as well as lower CDN costs.

1) CLS Overview: A typical CLS system is illustrated in Figure 10. After receiving viewers' requests, the CLS platform will first aggregate all stream data to the source server, and then deliver the them to viewers through CDN providers according to a certain scheduling strategy. This figure shows that viewer A, B, and C are watching broadcaster a, b,and c through CDN A, CDN B, CDN B, respectively.

Testbed Setup. As suggested by previous work [18], our experiments are conducted on the real-world CLS dataset provided by Kuaishou, spanning 1 week (6 days for training and 1 day for test). At each time, we select 3 candidates from total 4 different CDN providers, and we fit a separate simulator for them. We train Zwei for about 4 hours to converge in a stable result.

CDN Configuration. We detail the CDN configuration on Figure 9. Followed by previous work [18], we use a piecewise linear model to characterize this relationship between workload and block ratio (Figure 9(a)). Note that the following features are extracted by the real CDN dataset. What's more, we define the CDN pricing model w.r.t various CDN providers in industry, such as Amazon E2, and Tencent CDN, and plot the correlation between the workload and cost on Figure 9(b).

NN Representation. We implement Zwei in CLS as suggested by recent work [18]. More precisely speaking, for each CDN provider i, Zwei passes past 20 normalized workload and stalling ratio to a single CNN layer with filter=64, size=4, and stride=1. Then several output layers are merged into a hidden layer that uses 64 units to apply the *softmax* activation function. We model the action space as a heuristic way: each CDN provider has 3 choices instead: increase its configuration ratio by 1%, 5%, and 10% based on the previous stalling ratio. And Zwei will reduce the ratio for CDN that obtained worst previous ratio.

Requirements for CLS tasks. Algorithm 3 is an example of Rule `Rule` which used in the CLS scenario, where $\epsilon_c = 10$ is also a small number that can add noise to improve Zwei's robustness.

Algorithm 3 Rule for the CLS task.

```
Require: Trajectory $T_u, T_v$;
1: Compute underlying metric average stalling ratio $stall_u, stall_v$, accumulative cost $c_u, c_v$ from the given trajectories $T_u, T_v$.  ▷ Requirements: i) low stalling ratio ii) low costs.
2: Initialize Return $s = \{-1, -1\}$;
3: if $|stall_u - stall_v| < \epsilon$ or $|c_u - c_v| < \epsilon_c$ then
4:     Randomly set $s_0$ or $s_1$ as 1.  ▷ Add noise to improve the robustness.
5: else if $|stall_u - stall_v| < \epsilon_c$ then
6:     $s_i \leftarrow 1, i = argmin_{i \in \{u,v\}} c$;
7: else
8:     $s_i \leftarrow 1, i = argmin_{i \in \{u,v\}} stall$;
9: return $s$;
```

2) Baselines: Like other scenarios, we also compare Zwei with the following state-of-art scheduling baselines:

1) ***Weighted round-robin* [51].** The idea of weighted round-robin (WRR) is: the requests will be redirected to different CDN providers w.r.t a constant ratio. We adopt the algorithm with the *best* parameters.
2) ***E2* [6].** Exploitation and Exploration (E2) algorithm utilizes harmonic mean for estimating CDN providers' performance, and select with the highest upper confidence bound of reward. We use E2 algorithm provided by the authors [6].
3) ***LTS* [18].** State-of-the-art CLS algorithm which uses deep reinforcement learning to train the NN towards lower stalling ratio. However, it ignores the trade-off between the cost and the performance. We evaluate LTS by Zhang et. al. [18].

3) Zwei vs. State-of-the-art Scheduling Methods: We start to study how well that Zwei achieves under the CLS scenario. As shown in Figure 9(c), we find that Zwei stands for the best scheme among the candidates. Specifically, Zwei reduces the overall costs on 22% compared with state-of-the-art learning-based method LTS, and decreases over 6.5% in terms of the overall stalling ratio. The reason is that LTS takes the weight-sumed combination function as the reward, while the function can hardly give a clearer guidance for the optimized algorithm. Moreover, comparing the performance of Zwei with the optimal strategy generated by the reward function, we observe that both optimal policy and Zwei are in the Pareto front. Zwei comsumes less pricing costs than the optimal policy since the requirement is to *minimize the cost first*.

4) Experimental Results: Besides, we also present the training process in Figure 9(d). As shown, Zwei converges in less than 600 epochs, which needs about 3 hours. It's worth noting that Zwei also experienced two stages on the CLS task. The first stage ranges from 0 to 100 epochs, and we can see the goal of Zwei is to minimize the cost without considering the number of stalling ratio. The rest of the stage we find that Zwei attempts to reduce the number of stalling ratio, and at the same time, the cost is converge to a steady state. Such observation also proves that Zwei learns the strategies by following the given requirements.

C. RTC Tasks

To better understand how Zwei performs in the RTC task, we devElop a faithful packet-based network emulation testbed, train Zwei via various real-world network traces, and validate Zwei with scale. As expected, results also prove the superiority of Zwei against existing methods.

1) RTC Overview: We start with explaining the conventional end-to-end transmission process for real-time video streaming. As shown in Figure 11, the RTC system contains a sender and a receiver, and its transport protocol mainly consists of two channels: the streaming channel and the feedback message channel. At the beginning, the sender deploys a UDP socket channel to send the instant real-time video streaming packets $P = \{p_0, p_1, \cdots, p_k\}$, denoted as a packet train [52] to the receiver through the streaming channel. The receiver then feeds network status observed back to the sender through the feedback channel. Based on this information, the sender will select bitrate for next time period. Note that the feedback channel is usually assumed as a reliable channel since the transferred data in feedback channel per second is rather small.

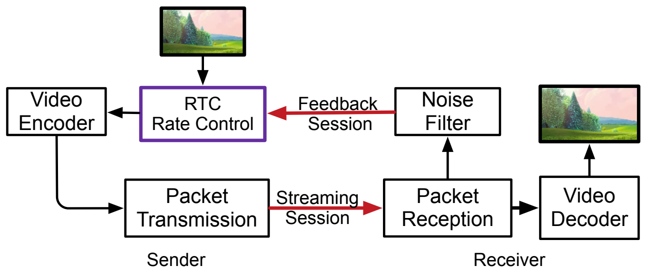


Fig. 11. RTC System Overview.

2) Network Simulator Setup: In this part, we aim to introduce the principle of the RTC network simulator. To train Zwei in the RTC scenario, we first consider training the neural network (NN) model in real-world network conditions, i.e., deploying the model on the edge server. The model will finally converge with the increased number of video session. However, it's hard to converge since online training should explore almost all network status. We then decide to train the model in simulated offline networks. Hence, we are facing a new challenge: *how to design a fast-forward network simulator which can precisely compute the latency with given saturated trace and sending rate?*

We adopt delay gradient instead of one-way delay to train Zwei. Delay gradient is defined as the difference on delay measurement on both side. [19] It is also proposed to measure the latency in un-synchronized clock environment. Delay gradient $q(t_i)$ is computed as Eq. 12, which is actually used to describe the variety of the queuing delay. If the network is congested, its delay gradient will be greater than zero, vice versa. If the delay gradient equals to zero, we can only infer that network is not congested.

$$q(t_i) = \frac{1}{N} \sum_{i=1}^{N} [(t_i - t_{i-1}) - (T_i - T_{i-1})] \tag{12}$$

Where T_i is the time at which the i-th packet has been sent, and t_i is the time at which the i-th packet has been received, and N represents the packet count in a period.

Our simulator should emulate the process of the packets coming and leaving in different network conditions, and keep track of the timestamps, by which we are able to obtain the corresponding queuing delay gradient. Inspired by recent work [53], we utilize saturated network trace to generate queuing delay data. As shown in Figure 12, assuming the distribution of packets arrival and leave fits closely to the Poisson process [53], we use sending bitrate and bandwidth in saturated network traces as the arriving rate λ and leaving rate μ, respectively.

In detail, we regard the channel link between the sender and the receiver as a single device. The device is comprised of three queues, namely *input*, *output* and *limbo*. An algorithm is designed to release packets from each queue corresponding to the network trace dataset in different levels. Each element in

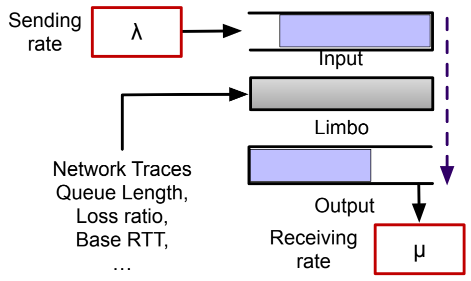


Fig. 12. The working principle of the network simulator.

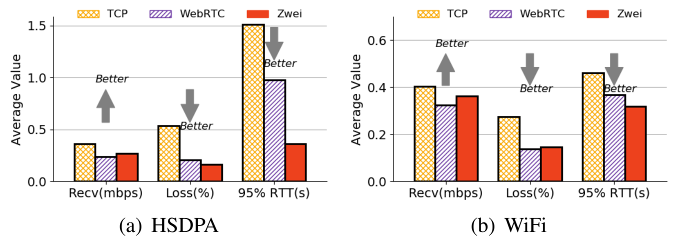


Fig. 13. The comparison of Zwei and WebRTC as well as TCP on both HSDPA and WiFi network datasets.

the queue is the opportunity of packet-delivery, which means, the time (in milliseconds) at which an MTU-sized packet will be delivered. For each period $(t_k, t_{k+1}]$, the offline-network simulator compares the timestamp of the front packet in *input* with that in *limbo*. If timestamp in *input* queue is bigger than or equal to the one in *limbo*, the simulator will push the front packet into the bottom of the *output* queue, and then computes the delay gradient between the two packets. On the contrast, the delivery opportunities will be wasted if the timestamp in *input* queue is smaller than that one in *limbo* queue. The simulator only pops the front packet in *limbo* queue. By this process, the simulator returns mean self-inflicted delay and total bytes that the sum of *output* queue every interval $t_{k+1} - t_k$.

3) NN Architecture Overview: We use past $k = 10$ sequence, and for each sequence we take past sending rate, past receive rate, delay gradient, round-trip time (RTT), loss ratio as the input. Besides, we output the NN as 11-dims vector a to control the sending rate, which represents {-0.4, -0.3, -0.2, -0.1, -0.05, 0.0, 0.05, 0.1, 0.2, 0.3, 0.4}. Inspired by the additive-increase, multiplicative-decrease (AIMD) algorithm, we take different logic to increase or decrease the sending rate. Zwei increases the sending rate r_t as $r_t = r_t + r_{max} \times a_t$, and reduces the sending rate as $r_t = r_t \times (1 - a_t)$. We set minimum sending rate $r_{min} = 0.01$ and $r_{max} = 0.5$ MB/s. The Zwei's NN representation is equal to previous work [9]. We use the combination of 1D-CNN and Fully connected layers to extract the feature and then merge into a vector. Finally, the NN outputs the vector w.r.t ABR tasks.

4) Implementation Details: Figure 11 elaborates RTC system overview. We place Zwei as the RTC control module on the sender-side. Followed by recently proposed approaches [9], we have to devElop a faithful network emulator which can simulate the network condition according to network saturate traces that collected from different network conditions. The network traces is composed of 3G, 4G/LTE and WiFi network as well as self-collected real-world network environments. Then we train Zwei under the simulator for about 8 hours and test it on the Mahimahi [40].

5) RTC Baselines: We compare Zwei with several proposed heuristic methods, including:

1) ***TCP* [5]**. The most popular congestion control algorithm in the Internet. The key insight is the additive-increase, multiplicative-decrease (AIMD) algorithm, which means a linear increasing of the congestion window with an exponential reduction when congestion is detected. We adopt TCP-Reno with *fast recovery*.
2) ***WebRTC* [19]**. State-of-art RTC scheme on both academic and industry. More precisely speaking, WebRTC leverage delay-based and loss-based module to jointly control the rate adaption problem in the real-time scenario, where its parameters have been carefully tuned for almost one decade. In this work, we adopt WebRTC from its open-sourced repository.

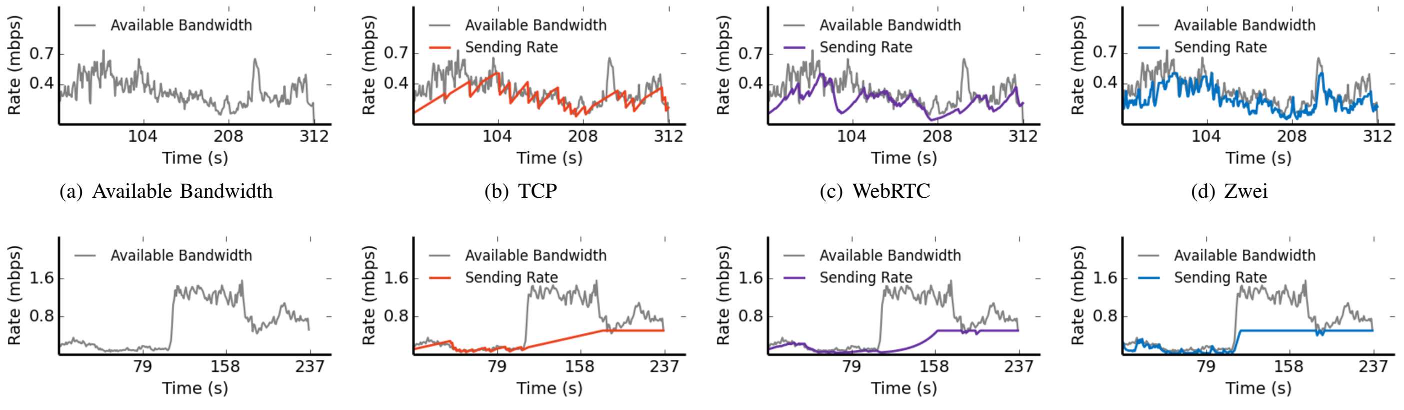
6) Zwei vs. RTC Baselines: In this experiment, we compare Zwei with existing proposed heuristic methods (§IV-C5) on various network conditions such as 3G and Wifi. We collect average receiving bitrate, 95 percent round-trip-time (RTT) as well as average loss ratio every 1 second. Note that these metrics are widely used in RTC evaluation [9]. Results are reported in Figure 13, we demonstrate that Zwei improves 11.34% on average receiving bitrate, reduces 20.49% in terms of loss ratio, and decreases 62.95% on 95-percent RTT compared with WebRTC under HSDPA dataset. Another results on the WiFi network conditions show that Zwei also outperforms WebRTC, with the improvements on sending rate of 12.08% and decreasing in 95% RTT of 13.59%. The comparison of Zwei and TCP illustrates that Zwei can effectively detect the congestion signal and dynamically adjust the sending rate to avoid high loss ratio and RTT.

7) Zwei Case Study: To better understand the performance of the selected methods, we **randomly** pick four traces from the validation dataset and compare the sending rate curve of Zwei with TCP and WebRTC. As demonstrated in Figure 14 and Figure 15, Zwei can accurately estimate the bottleneck of the session, as it always sends video streaming with high rate with avoiding congestion.

V. Related Work

A. Heuristic Methods

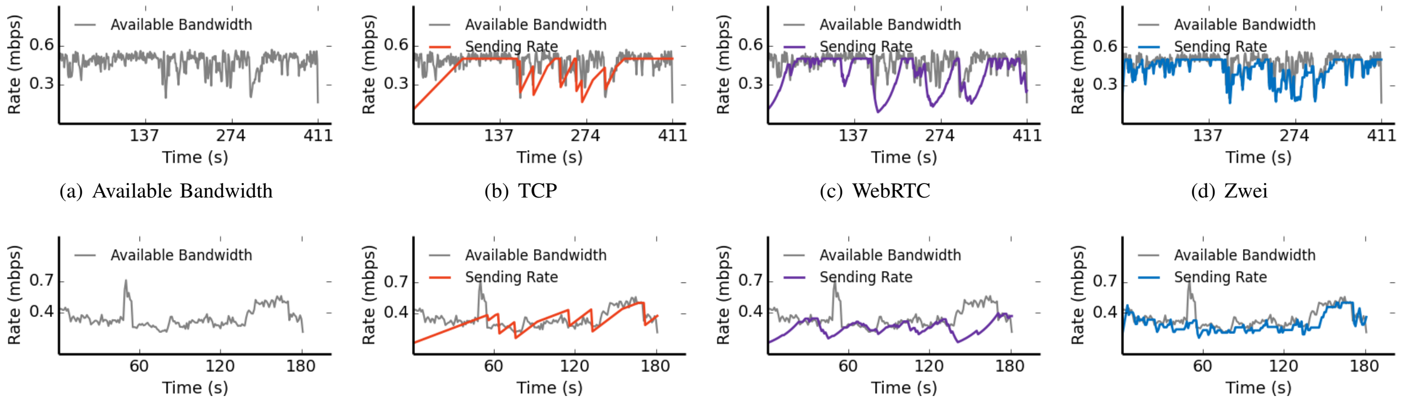
1) Adaptive Bitrate Video Streaming: Heuristic-based ABR methods often adopts throughput prediction (E.g., FESTIVE [37]) or buffer occupancy control (E.g., BOLA [23]) to handle the task. However, such approaches suffer from either inaccurate bandwidth estimation or long-term bandwidth fluctuation problems. Then, MPC [21] picks next chunks' bitrate by jointly considering throughput and buffer occupancy. Nevertheless, MPC is sensitive to its parameters since the it relies on well-understanding different network conditions. What's more, Oboe [10] even proposes an auto-tuning method



(a) Available Bandwidth (b) TCP (c) WebRTC (d) Zwei

(e) Available Bandwidth (f) TCP (g) WebRTC (h) Zwei

Fig. 14. The comparison of Zwei and existing methods on the HSDPA network trace.



(a) Available Bandwidth (b) TCP (c) WebRTC (d) Zwei

(e) Available Bandwidth (f) TCP (g) WebRTC (h) Zwei

Fig. 15. We also compare Zwei with existing methods on the trace collected from the real-world WiFi scenario.

to tune the traditional heuristic methods to achieve better performance in different network settings.

2) CSP Selection: Through preliminary measurements, it is widely accepted that the strategies are largely statically configured [51]. In recent years, dynamic scheduling across different CSPs has received more attention. Jiang et. al. [6] uses E2 method to replace traditional model-based methods.

3) RTC Rate Adaption: Rate adaption methods are mainly composed of loss-based approach [5], attempting to increase bitrate till packet loss occurs, and delay-based approach [54], aiming to adjust sending rate to control the transmission delay. However, such methods are sensitive to network conditions. Thus, model-based approach, such as WebRTC [19], controls the bitrate based on previous status observed, including past sending rate, receiving rate, delay gradients, as well as loss ratio. In short, heuristic-based methods require careful tuning their parameters.

B. Learning-based Schemes

Recent years, several learning-based attempts have been made to tackle video transmission problem. For example, Mao et al. [8] develop an ABR algorithm that uses DRL to select next chunks' bitrate. D-DASH [55] uses Deep Q-learning method to perform a comprehensive evaluation based on state-of-the-art algorithms. Tiyuntsong optimizes itself towards a rule or a specific reward via the competition with two agents under the same network condition [11]. LTS [18] is a DRL-based scheduling approach which outperforms previously proposed CLS approaches. Furthermore, QARC [9] optimizes a NN towards higher perceptual video quality and less stalling events. However, such methods fail to achieve actual requirements since they are optimized via a linear-based reward function.

VI. Conclusion

We propose Zwei, which utilize the comparison results between two samples to enhance itself towards a better *win rate*. Zwei can generalize an outstanding strategy based on the actual requirement, where the requirement is always hard to be described as a linear-based manner. Results show that Zwei outperforms recent state-of-the-art work with the improvements of more than 22% in terms of three fundamental video transmission tasks.

References

[1] Cisco, "Cisco visual networking index: Forecast and methodology, 2016-2021," 2017. [Online]. Available: https://www.cisco.com/c/dam/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.pdf

[2] R. Rejaie, M. Handley, D. Estrin *et al.*, "Quality adaptation for congestion controlled video playback over the internet," in *SIGCOMM'99*, 1999.

[3] C. Lefelhocz, B. Lyles, S. Shenker, and L. Zhang, "Congestion control for best-effort service: why we need a new paradigm," *IEEE Network*, vol. 10, no. 1, pp. 10–19, 1996.

[4] T.-Y. Huang, R. Johari, N. McKeown, M. Trunnell, and M. Watson, "A buffer-based approach to rate adaptation: Evidence from a large video streaming service," *SIGCOMM'14*, vol. 44, no. 4, 2015.

[5] J. Mo, R. J. La, V. Anantharam, and J. Walrand, "Analysis and comparison of tcp reno and vegas," in *INFOCOM'99*, vol. 3, 1999.

[6] J. Jiang, S. Sun, V. Sekar, and H. Zhang, "Pytheas: Enabling data-driven quality of experience optimization using group-based exploration-exploitation." in *NSDI*, vol. 1, 2017.

[7] H. Mao, S. B. Venkatakrishnan, M. Schwarzkopf, and M. Alizadeh, "Variance reduction for reinforcement learning in input-driven environments," *ICLR'19*, 2019.

[8] 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.

[9] T. Huang, R.-X. Zhang, C. Zhou, and L. Sun, "Qarc: Video quality aware rate control for real-time video streaming based on deep reinforcement learning," in *Multimedia'18*, 2018.

[10] Z. Akhtar, P. Adria, M. Mirza *et al.*, "Oboe: auto-tuning video abr algorithms to network conditions," in *SIGCOMM 2018*, 2018.

[11] T. Huang, X. Yao, C. Wu, R.-X. Zhang, and L. Sun, "Tiyuntsong: A self-play reinforcement learning approach for abr video streaming," *arXiv preprint arXiv:1811.06166*, 2018.

[12] T.-Y. Huang, C. Ekanadham, A. J. Berglund, and Z. Li, "Hindsight: Evaluate video bitrate adaptation at scale," in *MMSys'19*, ser. MMSys '19. New York, NY, USA: ACM, 2019, pp. 86–97. [Online]. Available: http://doi.acm.org/10.1145/3304109.3306219

[13] H. Mao, S. Chen, D. Dimmery, S. Singh, D. Blaisdell, Y. Tian, M. Alizadeh, and E. Bakshy, "Real-world video adaptation with reinforcement learning," 2019.

[14] D. Silver, T. Hubert, J. Schrittwieser *et al.*, "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," *CoRR*, vol. abs/1712.01815, 2017. [Online]. Available: http://arxiv.org/abs/1712.01815

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

[16] 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 *ICML'17*, 2016.

[17] A. Elo, *The rating of chessplayers, past and present*. Arco Pub., 1978. [Online]. Available: https://books.google.com/books?id=8pMnAQAAMAAJ

[18] R.-X. Zhang, T. Huang, M. Ma, H. Pang, X. Yao, C. Wu, and L. Sun, "Enhancing the crowdsourced live streaming: a deep reinforcement learning approach," in *NOSSDAV'19*, 2019.

[19] G. Carlucci, L. De Cicco, S. Holmer, and S. Mascolo, "Analysis and design of the google congestion control for web real-time communication (webrtc)," in *MMSys'16*, 2016.

[20] A. Bentaleb, B. Taani, A. C. Begen, C. Timmerer, and R. Zimmermann, "A survey on bitrate adaptation schemes for streaming media over http," *IEEE Communications Surveys & Tutorials*, 2018.

[21] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli, "A control-theoretic approach for dynamic adaptive video streaming over http," in *SIGCOMM'15*, 2015.

[22] Y. Sun and et al., "Cs2p: Improving video bitrate selection and adaptation with data-driven throughput prediction," in *SIGCOMM 2016*, 2016.

[23] K. Spiteri *et al.*, "Bola: Near-optimal bitrate adaptation for online videos," in *INFOCOM'16*, 2016.

[24] H. Riiser, P. Vigmostad, C. Griwodz, and P. Halvorsen, "Commute path bandwidth traces from 3g networks: analysis and applications," in *Proceedings of the 4th ACM Multimedia Systems Conference*, 2013.

[25] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel *et al.*, "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," *arXiv preprint arXiv:1712.01815*, 2017.

[26] G. Tesauro and G. R. Galperin, "On-line policy improvement using monte-carlo search," in *Advances in Neural Information Processing Systems*, 1997, pp. 1068–1074.

[27] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International conference on machine learning*, 2015, pp. 1889–1897.

[28] Y. Wu, E. Mansimov, R. B. Grosse, S. Liao, and J. Ba, "Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation," in *Advances in neural information processing systems*, 2017.

[29] T. Huang, "Zwei," https://github.com/godka/Zwei, 2020.

[30] Y. Tang, "Tf. learn: Tensorflow's high-level module for distributed machine learning," *arXiv preprint arXiv:1612.04251*, 2016.

[31] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: A system for large-scale machine learning." in *OSDI*, vol. 16, 2016.

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

[33] "Http live streaming," https://developer.apple.com/streaming/, 2019.

[34] "Dash industry forum — catalyzing the adoption of mpeg-dash," 2019. [Online]. Available: https://dashif.org/

[35] T. Huang, C. Zhou, R.-X. Zhang, C. Wu, X. Yao, and L. Sun, "Comyco: Quality-aware adaptive video streaming via imitation learning," *arXiv preprint arXiv:1908.02270*, 2019.

[36] P. G. Pereira, A. Schmidt, and T. Herfet, "Cross-layer effects on training neural algorithms for video streaming," in *Proceedings of the 28th ACM SIGMM Workshop on Network and Operating Systems Support for Digital Audio and Video*, 2018.

[37] J. Jiang, V. Sekar, and H. Zhang, "Improving fairness, efficiency, and stability in http-based adaptive video streaming with festive," *TON*, vol. 22, no. 1, 2014.

[38] K. Spiteri, R. Sitaraman, and D. Sparacio, "From theory to practice: improving bitrate adaptation in the dash reference player," in *Proceedings of the 9th MMSys*, 2018.

[39] Mao, "hongzimao/pensieve," Jul 2017. [Online]. Available: https://github.com/hongzimao/pensieve

[40] R. Netravali, A. Sivaraman, S. Das, A. Goyal, K. Winstein, J. Mickens, and H. Balakrishnan, "Mahimahi: Accurate record-and-replay for {HTTP}," in *ATC'15)*, 2015.

[41] H. Mao, P. Negi, A. Narayan, H. Wang, J. Yang, H. Wang, R. Marcus, R. Addanki, M. Khani, S. He *et al.*, "Park: An open platform for learning augmented computer systems," 2019.

[42] M. F. B. Report, "Raw data measuring broadband america 2016," https://www.fcc.gov/reports-research/reports/measuring-broadband-america/raw-data-measuring-broadband-america-2016, 2016, [Online; accessed 19-July-2016].

[43] "Big buck bunny," Jan 2020. [Online]. Available: http://www.bigbuckbunny.org

[44] A. Dethise, M. Canini, and S. Kandula, "Cracking open the black box: What observations can tell us about reinforcement learning agents," in *Proceedings of the 2019 Workshop on Network Meets AI & ML*, 2019.

[45] R. Coulom, "Whole-history rating: A bayesian rating system for players of time-varying strength," in *International Conference on Computers and Games*. Springer, 2008, pp. 113–124.

[46] D. Silver, A. Huang, C. J. Maddison *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, 2016.

[47] "Pensieve-ppo," Jan 2020. [Online]. Available: https://github.com/godka/Pensieve-PPO

[48] I. J. Goodfellow, M. Mirza, D. Xiao, A. Courville, and Y. Bengio, "An empirical investigation of catastrophic forgetting in gradient-based neural networks," *arXiv preprint arXiv:1312.6211*, 2013.

[49] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[50] A. Haghighat and J. C. Wagner, "Monte carlo variance reduction with deterministic importance functions," *Progress in Nuclear Energy*, vol. 42, no. 1, pp. 25–53, 2003.

[51] V. K. Adhikari, Y. Guo, F. Hao, M. Varvello, V. Hilt, M. Steiner, and Z.-L. Zhang, "Unreeling netflix: Understanding and improving multi-cdn movie delivery," in *INFOCOM'12*, 2012.

[52] N. Sato, T. Oshiba, K. Nogami, A. Sawabe, and K. Satoda, "Experimental comparison of machine learning-based available bandwidth estimation methods over operational lte networks," in *Computers and Communications (ISCC), 2017 IEEE Symposium on*, 2017.

[53] K. Winstein, A. Sivaraman, and H. Balakrishnan, "Stochastic forecasts achieve high throughput and low delay over cellular networks," 2013.

[54] D. Rossi *et al.*, "Ledbat: The new bittorrent congestion control protocol." in *ICCCN*, 2010.

[55] 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, Dec 2017.