*electronics*

MDPI

*Article*

# PaCs: Playing Time-Aware Chunk Selection in Short Video Preloading

**Sen Fu [1,2], Guanyu Yang [3], Zhengjun Yao [1], Shuxin Tan [4,*], Yongxin Shan [5] and Wanchun Jiang [5]**

[1] College of Materials Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 211106, China; fs211314@gmail.com (S.F.); yaozj@nuaa.edu.cn (Z.Y.)
[2] China Aerospace Science and Industry Corporation, Beijing 100074, China
[3] Sino-French Engineer School, Beihang University (BUAA), Beijing 100191, China; 18241048@buaa.edu.cn
[4] School of Literature and Communication, Hunan Women's University, Changsha 410116, China
[5] School of Computer Science and Engineering, Central South University, Changsha 410075, China
* Correspondence: tanyatsx92@163.com

**Abstract:** Short video applications are popular nowadays. Typically, a short video may be played for a few seconds or swiped away by users at any time. Due to this uncertain user behavior, video chunks should be preloaded to ensure a smooth viewing process for users. In other words, the short video preloading scheme is crucial to the quality of experience (QoE) of users and revenue. Specifically, the short video preloading scheme should determine which video chunk to download, when to download it, and at what bitrate. Existing schemes either fail to consider all these factors together or find it hard to make the best decision. In this work, we argue that the selection of downloaded video chunks is the foremost task due to the uncertain user behaviors and the corresponding huge number of possible playing sequences of video chunks. As a step further, we propose the playing time-aware chunk selection (PaCs) scheme, which downloads the video chunk with the smallest expected playing time first. After the selection of the video chunk, the bitrate is selected according to the classic MPC algorithm and then whether the downloading process is paused or executed is discussed. In total, PaCs can improve the score consisting of the QoE of downloaded video chunks and the utilized network bandwidth under different conditions. Simulations confirm that PaCs achieves a higher score than the existing scheme Dashlet proposed in NSDI 2024.

**Keywords:** short video; preloading; bitrate selection; video chunk selection; pause

check for updates

**Citation:** Fu, S.; Yang, G.; Yao, Z.; Tan, S.; Shan, Y.; Jiang, W. PaCs: Playing Time- Aware Chunk Selection in Short Video Preloading. *Electronics* **2024**, *13*, 4864. https://doi.org/10.3390/electronics13244864

Academic Editor: Andreas Mauthe

Received: 22 October 2024
Revised: 26 November 2024
Accepted: 28 November 2024
Published: 10 December 2024

**Copyright:** © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

## 1. Introduction

Short video applications possess strong user appeal and interactivity and short video platforms such as TikTok, Instagram Reels, and Kuaishou have hundreds of millions of users worldwide [1]. For instance, TikTok has amassed almost 5.7 billion global downloads since its release on 5 August 2024 [2] and has 3.065 billion monthly active users as of April 2024 [3]. In short video applications, users can either watch the next chunk of the current video or switch to the next video at any time. To improve the quality of experience (QoE) of users, applications preload video chunks before they are watched. In other words, the short video preloading scheme determines which video chunk to download next, when to download it, and what bitrate is crucial to the applications.

Many works focus on enhancing the performance of the short video preloading scheme under different scenarios. Specifically, adaptive bitrate (ABR) algorithms are designed for on-demand or live video streaming and solely focus on the bitrate selection according to variable network bandwidth. Ignoring the questions of which video chunk to download next and when to download it, ABR algorithms would perform poorly in short video scenarios with uncertain user behaviors. In a short video scenario, TikTok preloads a fixed number of chunks for each video in order, ignoring the impacts of the uncertain user behavior. To improve the performance of short video preloading, the PDAS scheme,

which achieved the highest score in the Short Video Streaming Grand Challenge in ACM Multimedia 2022, employs the user retention probability (RP) to reflect uncertain user behavior and considers the selection of both video chunks and their bitrates. Similarly, the Dashlet scheme proposed in NSDI 2024 also considers these factors and sorts chunks to be downloaded by minimizing the expected rebuffering times. Although they may achieve good performance, the computation complexity is large due to the association between video chunk selection and bitrate selection.

In this work, we present a playing time-aware chunk selection (PaCs) scheme. To reduce the computation complexity, PaCs makes video chunk selection according to the user retention probability, ahead of bitrate selection according to the variable network bandwidth. Specifically, PaCs predetermines the relative order of candidate chunks for downloading based on application playback constraints at first. This constraint means that later chunks in a video are only reachable via earlier ones, and later videos are only reachable via swipes from earlier ones. Secondly, PaCs calculates the expected playing time for each potential chunk based on the aforementioned inter-chunk relationships and the user retention probability. Finally, PaCs sorts video chunks to be downloaded as close as possible to the playing order and obtains the recommended download sequence of the chunks within the current time horizon based on the expected playing time. After the above video chunk selection, the traditional ABR algorithm model predictive control (MPC) is employed to determine the bitrates of the recommended download sequence of chunks according to the estimated network bandwidth. The first video chunk in this sequence with the selected bitrate will be downloaded. Instead of directly downloading the selected video chunk with the selected bitrate, PaCs also considers whether the downloading should be paused.

To evaluate the performance of PaCs, we implement it in the short video simulator for the Short Video Streaming Grand Challenge in ACM Multimedia 2022 [4]. The results show that PaCs improves the score consisting of the quality of experience (QoE) of downloaded video chunks and the utilized network bandwidth under different conditions. Specifically, compared with TikTok, PaCs improves the Score by 28%, 29%, and 39% in low, medium, and high network bandwidth, respectively. Moreover, PaCs also achieves a higher score than PDAS, which won the first prize in the ACM Multimedia 2022, and Dashlet, proposed in NSDI 2024.

The rest of this paper is organized as follows. Section 2 provides the background and related work, and Section 3 is the design of PaCs. Section 4 is the evaluation of PaCs, and Section 5 concludes this paper.

## 2. Background and Related Work

### *2.1. Short Video System*

As shown in Figure 1, the short video system consists of clients and servers. On the client side, the short video application is employed. In the short video application, the video playback module refers to the screen, where the video chunks are either played or swiped away by users. Typically, many video chunks are preloaded in the buffer to avoid rebuffering. Moreover, because the current video can be swiped away at any time, the downloaded chunks belong to both the current video and the other recommended videos, as illustrated in Figure 1. Furthermore, the short video preloading scheme module is responsible for the video chunk and bitrate selection, i.e., determines which video chunk to download next, when to download it, and at what bitrate. Note that the candidate video chunks for selection are all the subsequent chunks of all the videos in the buffer. On the server side, the HTTP request is received from clients, and the corresponding video chunk assigned in the HTTP request will be sent to the clients. Typically, the server refers to the content delivery network (CDN), where the video chunks have been distributed. In total, the short video system maintains a reasonable buffer of video chunks such that the likelihood of rebuffering is reduced in the face of the dynamic bandwidth and the uncertain users swiping among different short videos.

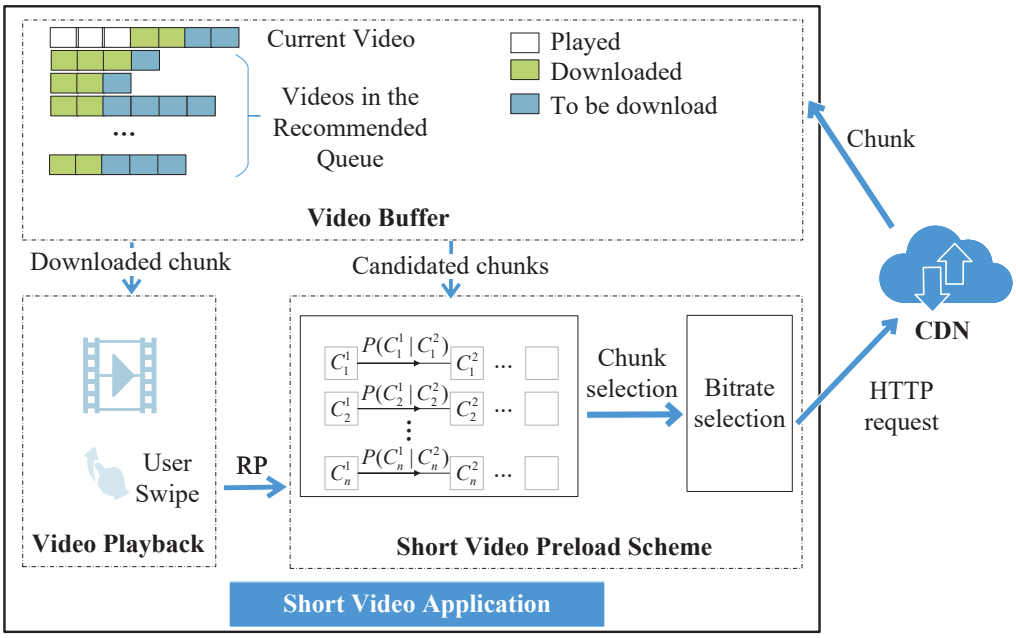


**Figure 1.** The short video system.

### *2.2. Short Video Preloading*

Excepting the bitrate selection as the classic ABR algorithm, the short video preloading scheme should also be used to determine which video chunk to download next and when to download it due to the uncertain user behavior. Correspondingly, the performance metric of the short video preloading scheme consists of the following two parts. The first one is the QoE of the played video chunk, similar to ABR. The difference is that these chunks may belong to different videos. Specifically, QoE is always the weighted sum of the bitrate, the smoothness, and the rebuffering time of the played video chunks. In other words, the QoE of video $\epsilon$ can be mathematically described by

$$QoE_\epsilon = \sum_l (\beta \times b_l - \theta \times |b_l - b_{l-1}|) - \sum_l \gamma \times r_l \quad (1)$$

where for chunk $l$ of video $\epsilon$, $b_l$ is the bitrate of chunk $l$, and $r_l$ refers to the rebuffering time of playing video chunk $l$. The smoothness is represented by $|b_l - b_{l-1}|$, i.e., bitrate switching. Moreover, $\beta$, $\theta$, and $\gamma$ are constant weight parameters, similar to [5]. More specifically, bitrate directly reflects the clarity and quality of the video chunks and it should be as high as possible. Moreover, smoothness refers to dynamic bitrate switching during video playback, and frequent bitrate switching would affect user experience. Furthermore, rebuffering time occurs when there are no video chunks to play, and it negatively influences the user experience. For example, the current short video may probably be swiped away in the face of large rebuffering time. The second one is bandwidth usage, which refers to the network capacity consumed for downloading video chunks. This is because some of the downloaded video chunks will not be played when users swipe away from the corresponding video. Accordingly, the bandwidth capacity used to download these video chunks is wasted. In total, let $Usage_\epsilon$ represent the bandwidth consumed by downloading video $\epsilon$, the performance of the short video preloading scheme can be evaluated by the following score [4].

$$Score = \sum_\epsilon (QoE_\epsilon - \alpha \times Usage_\epsilon) \quad (2)$$

where $\alpha$ is a constant weight parameter.

With the above definition, the performance of the short video preloading scheme is greatly influenced by the following factors. The first one is the uncertain user behaviors.

On the one hand, several videos in the recommended queue should be preloaded to avoid rebuffering in case the current video is swiped away. Accordingly, chunk selection becomes one of the core tasks of the short video preloading scheme, except for the bitrate selection. However, chunk selection is challenging because the user may swipe from any video chunk to another video. On the other hand, the buffered chunks of the current video, which were downloaded previously, would no longer be played after swiping to another video, leading to bandwidth wastage. Therefore, the number of buffered chunks should not be so large as to increase the bandwidth usage. In addition, pausing may also be a better decision than selecting a chunk to download, as the bandwidth usage may be greatly improved by the pause decision. The second factor is the fluctuating network bandwidth. In the face of the unknown network bandwidth for the subsequent chunk downloading, the bitrate selection is crucial to the QoE, as explored by the numerous works on ABR algorithms [6–8]. Meanwhile, the unknown network bandwidth also determines whether the selected video chunk can be downloaded in time, i.e., before this chunk is played or the rebuffering occurs.

In sum, the short video preloading scheme should select a proper video chunk, select a proper bitrate for it, and decide the time of downloading it under uncertain user behaviors and the fluctuating network bandwidth. The goal is to maximize the score represented by the weighted sum of QoE and the bandwidth usage.

### *2.3. Related Work*

The study of short video preloading schemes has drawn a lot of attention recently. On the one hand, ABR algorithms [6–14] are not directly applicable to short video preloading. Specifically, ABR algorithms designed for on-demand video and live video do not preload chunks of the next recommended video in advance. Accordingly, the rebuffering time would be large when users swipe away from the current video. On the other hand, several short video preloading schemes have been proposed. By default, TikTok preloads a fixed number of chunks for each video in order. In this way, the negative impacts of the uncertain user behavior are ignored. To address this issue, several short video preloading schemes [15–19], which conduct chunk selection according to uncertain user behavior, are proposed. Specifically, APL [19] predicts the playing time of video chunks based on the past playing behavior of users and suggests taking just the next chunk of each video in the recommended queue into consideration. In this way, the chunk selection problem is formulated as an online minimization of the rebuffering time problem and solved by the Lyapunov optimization theory. However, APL ignores the bitrate adaptation. Differently from APL, work [15,17] and PDAS [16] determine which video chunk to download based on whether the corresponding video buffer occupancy has reached the adaptive threshold. Moreover, the adaptive threshold is set based on the estimated bandwidth, chunk size, and given user retention probability (RP), i.e., the played probability of each chunk in the video. Here, RP is pre-defined according to the collected data about the playing times of video chunks for numerous users and is supposed to reflect uncertain user behaviors [4]. However, the adaptive threshold is set for each video independently, but users may swipe across different videos. In contrast, it is very complex to set the adaptive threshold for all videos in the recommended queue. To reduce complexity, Dashlet [18], which was published recently in NSDI, sorts the candidate video chunks according to their estimated rebuffering time by assuming that all chunks are downloaded at the maximum bitrate. After sorting video chunks, Dashlet selects bitrate based on the MPC algorithm [6]. The issue with Dashlet is that the bitrates of downloaded chunks are inconsistent with the bitrates used for sorting chunks. Accordingly, the performance of Dashlet may not be optimal, as discussed below.

On the other hand, being aware of the large complexity, several learning-based short video preloading schemes [5,20–26] have also been proposed recently. Although the learning-based schemes may have good performance, they generally have the issues of slow convergence and difficult deployment.

### *2.4. Dashlet*

Subsequently, the brief of Dashlet is introduced because the core idea of both Dashlet and PaCs is to make chunk selection ahead of bitrate selection so that the impact of uncertain user behavior is fully considered.

In detail, Dashlet calculates the rebuffering time for each candidate chunk. At first, Dashlet assumes that the playing time of each chunk is 5 s and selects all candidate video chunk that are possibly played within 25 s. In other words, 5 s chunks of each video are the candidates of Dashlet. Each permutation of these video chunks is a possible playback path in the future. For each video chunk in this playback path, the start time of playing it can be calculated. In addition, the probability that a playback path is followed can also be calculated according to the given user retention probability.

Subsequently, Dashlet considers the downloading time of each candidate video chunk and its impacts on the rebuffering time. To reduce the complexity, Dashlet assumes that all the candidate chunks are downloaded with the maximum bitrate. Accordingly, the size of the chunk can be obtained after knowing the bitrate, and the download time of the chunk can be calculated based on the estimated network bandwidth. As a step further, the rebuffering time of each chunk in each playback path can be calculated.

Next, Dashlet sorts all the candidate chunks according to the rebuffering time. More specifically, for any two candidate chunks, Dashlet calculates how much the rebuffering time can be reduced by exchanging the position of these two chunks in every playback path. Meanwhile, the probability of following the corresponding playback path is assigned as the weight of the rebuffering time. If the weighted sum of the rebuffering time of all playback paths is reduced, the position of these two chunks will be exchanged. After all the candidate chunks are sorted, Dashlet selects the bitrate according to the MPC algorithm.

As confirmed in [18], Dashlet greatly improves the performance of short video preloading. However, the maximum bitrate used for the calculation of the downloading time may be different from the final bitrate selected according to the MPC algorithm. This discrepancy of bitrates would hurt the performance of Dashlet, as we will show in this paper. Specifically, when the network bandwidth is high, MPC probably selects the highest bitrate and the discrepancy disappears. In contrast, when the network bandwidth is low, the discrepancy would be significant, and Dashlet performs poor. Additionally, after obtaining the candidate chunks, Dashlet applies a greedy algorithm to sort the chunks based on minimizing rebuffering time, resulting in significant computational overhead.

## 3. Design of PaCs

### *3.1. Framework*

The core idea of designing PaCs is to sort video chunks within a limited time scale according to their expected playing time. Specifically, as shown in Figure 2, a fixed time scale is predefined. Video chunks that may be played within this time scale are selected, and all the permutations of these video chunks, i.e., all the possible playback paths, are considered as follows. On the one hand, similar to Dashlet, PaCs can calculate the probability of each playback path being followed by users based on the given user retention probability. On the other hand, different from Dashlet, PaCs calculates the expected playing time for each video chunk without assuming the bitrate. In total, the expected playing time of each video chunk is the sum of the product of its playing time in each playback path and the probability of this playback path. According to the expected playing time, PaCs sorts all these video chunks as the final candidate sequence.

Subsequently, PaCs would make bitrate selection for the candidate sequence based on the MPC algorithm [6]. Finally, PaCs will check whether the overhead of the bandwidth usage is large enough, i.e., whether pausing is a better decision than downloading the first video chunk with the selected bitrate in the candidate sequence of video chunks. If the first chunk of the candidate sequence is the better choice, this chunk will be requested from the CDN server, as shown in Figure 2. Otherwise, the decision of the short video preloading scheme would be the Pause.

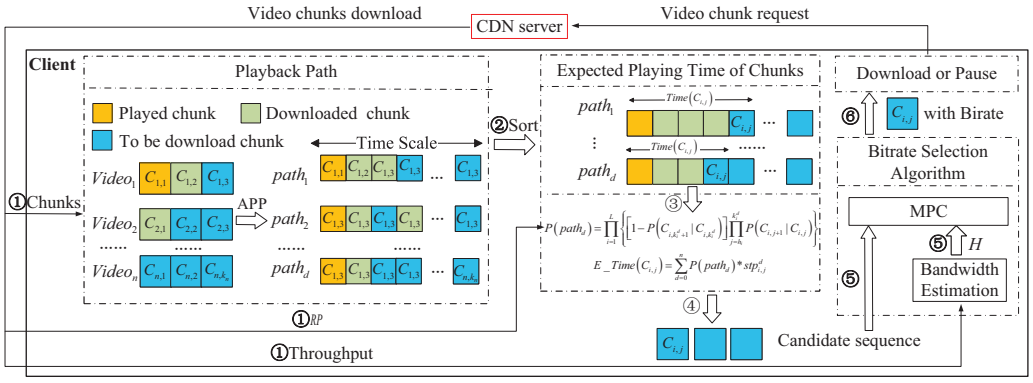


**Figure 2.** Framework of PaCs.

### *3.2. Detail of PaCs*

Similar to Dashlet, PaCs assumes that the playing time of each video chunk is $T$ seconds and considers all the video chunks, which will be played within time $T_s$. Each permutation of these video chunks is a playback path. $path_d$ represents the $d$-th possible playback path. Let $L$ denote the number of videos and $C_{i,j}$ represent the video chunk. Here, $i$ is the video index and $j$ is the chunk index. Let $h_i$ denote the chunk currently being played in video $i$. Obviously, $h_i = 1$, when $i > 1$. Let $k_i^d$ denote the final played chunks of video $i$ in $path_d$. Moreover, after chunk $C_{i,k_i^d}$ of the $i$-th video is played, the first chunk of the next video $C_{i+1,1}$ will be played. With the above definition, $path_d$ can be described as follows.

$$path_d = [C_{1,h_1}, \cdots, C_{i,k_i^d}, C_{i+1,1}, \cdots, C_{L,k_L^d}] \tag{3}$$

Accordingly, the number of chunks $D_{i,j}^d$ that have been played before video chunk $C_{i,j}$ in $path_d$, is

$$D_{i,j}^d = \sum_{m=1}^{i-1} (k_m^d - h_m + 1) + (j - h_i + 1) \tag{4}$$

Hence , the start play time $stp_{i,j}^d$ of playing video chunk $C_{i,j}$ in $path_d$ is

$$stp_{i,j}^d = T \times D_{i,j}^d \tag{5}$$

In addition, PaCs assumes that the user retention probability can be obtained, similar to works [8,16,27]. More specifically, the probability of playing video chunk $C_{i,j+1}$ or $C_{i+1,1}$ after playing $C_{i,j}$ can be obtained through the huge number of data collected in short video applications in history. If $P(C_{i,j+1}|C_{i,j})$ represent the probability that the user continues to watch video $i$, the probability that the user follows the $path_d$ can be calculated by

$$P(path_d) = \prod_{i=1}^{L} \{[1 - P(C_{i,k_i^d+1}|C_{i,k_i^d})] \prod_{j=h_i}^{k_i^d} P(C_{i,j+1}|C_{i,j})\} \tag{6}$$

Here , $C_{i,j+1}$ and $C_{i,j}$ represent two chunks connected; $1 - P(C_{i,k_i^d+1}|C_{i,k_i^d})$ represents the transition probability that users watch the next video $i+1$ after finishing watching chunk $C_{i,k_i^d}$ based on the user retention probability.

As shown in Algorithm 1, let $n$ denote the number of permutations of all the video chunks played within time scale $T_s$, there are $n$ playback paths

$$Paths(T_s) = [path_1, path_2, \cdots, path_d, \cdots, path_n] \tag{7}$$

and there must be $\sum_{d=1}^{n} P(path_d) = 1$.

**Algorithm 1** The PaCs algorithm

**Require:** $past_throughput, chunk_size, buffer, T_s, RP$//①
Play_paths = Paths($T_s$) //② *Equation* (7)
**for** path **in** Play_paths:
  Probs[path] = P(path,RP) // *Equation* (6)
Chunks = get_chunk(Play_paths) //③ *Chunks that appear in the playback path*
**for** i **in** len(Chunks):
  **for** j **in** len(Chunks[i]):
    **for** path **in** Play_paths:
      **if** Chunks[i][j] **in** path:
        E_Time[i][j] += Probs[path] * stp(chunk[i][j]) *//Equations* (5) *and* (8)
      **else**:
        E_Time[i][j] += Probs[path] * $T_s$
*//Equation* (8)
Candidate_seq = Sort(Chunks,E_Time,buffer) //④ *Sort the chunks not downloaded based on their expected playing time*
Seq = MPC(Candidate_seq, past_throughput, buffer, chunk_size) //⑤ *Get the bitrate of sequence through MPC*
Pause_Seq = pause + Seq *//Add a pause decision at the beginning of the sequence*
**if** E(Pause_Seq) > E(Seq): //⑥ *Check whether the download can be paused. Equation* (9)
  **return** Seq[0] *//Select the first chunk and its bitrate*
**else**:
  **return** pause *//Pause the download*

Subsequently, PaCs calculates the expected playing time for each video chunk $C_{i,j}$, which is possible to be played within the given time scale $T_s$. Referring to Equations (5) and (6), the expected playing time can be calculated as

$$E_Time(C_{i,j}) = \sum_{d=0}^{n} P(path_d) \times stp_{i,j}^{d} \tag{8}$$

Note that if video chunk $C_{i,j}$ is not involved in the $path_d$, the value of $stp_{i,j}^{d}$ is set as $T_s$. With the expected playing time, these video chunks are sorted, and the chunks that have not been downloaded make up the candidate sequence of PaCs. In this way, the sequence of downloading video chunks is expected to be consistent with the sequence of playing video chunks as much as possible.

After obtaining the candidate sequence by sorting video chunks with their expected playing time, PaCs makes bitrate selection according to the MPC algorithm, the same as Dashlet. In this way, the QoE is expected to be improved by adapting to the dynamic bandwidth.

Finally, PaCs decides whether to download the video chunks in the candidate sequence or to pause, thereby reducing bandwidth wastage. In detail, let *Seq* denote the sequence of video chunks, which are selected to be downloaded and assigned a bitrate in the candidate sequence. Meanwhile, let *Pause_Seq* denote the decision sequence, where a Pause decision is added at the beginning of *Seq*. The expected rebuffering time of sequences can be calculated as follows.

$$\begin{aligned} E(Seq) = \sum_{l \in Seq} \sum_{d=0}^{n} [P(path_d) \times \\ max\{std_{i_l,j_l} + dt_{i_l,j_l} - stp_{i_l,j_l}^{d}, 0\}] \end{aligned} \tag{9}$$

Here , $i_l$ and $j_l$ represent the video index and chunk index of the video chunk downloaded by decision $l$. $P(path_d)$ refers to the probability that the user follows. $stp_{i_l,j_l}^{d}$ is the start playing time of chunk $C_{i_l,j_l}$ defined in Equation (5). Moreover, $dt_{i_l,j_l}$ is the download time of chunk $C_{i_l,j_l}$, and it can be calculated based on the estimated network bandwidth and assigned bitrate. Furthermore, $std_{i_l,j_l}$ is the starting download time of chunk $C_{i_l,j_l}$, and it is

equal to the accumulated download time of the chunks before chunk $C_{i_l,j_l}$, which is similar to Dashlet, as described in Section 2.4. By default, the time of pausing the downloading is 0.1 s, i.e., the $std_{i_l,j_l}$ of $Pause_Seq$ pause is 0.1 s later than the $std_{i_l,j_l}$ of $Seq$. As a step further, if $E(Pause_Seq)$ is greater than $E(Seq)$, it means that the pause decision would increase the rebuffering time. Under this condition, PaCs would pause the downloading and request the first video chunk of the candidate sequence with the assigned bitrate from the CDN server. Otherwise, it indicates that the chunk can still be fully downloaded before playing even if the download is paused for a while. Under this condition, PaCs chooses to pause the download for 0.1 s such that the bandwdith would not be wasted.

### *3.3. Discussion*

Compared with other short video preloading schemes, PaCs decouples the selection of chunk preload order from bitrate selection, thus fully considering the impact of uncertain user behavior. PaCs can make reasonable chunk selection decisions in time and thus achieve good performance. Similarly, Dashlet also makes chunk selection ahead of bitrate selection. Based on the principle of minimizing the rebuffering time, the chunks are sorted. However, its chunk selection still relies on the assumption that the downloading time of the chunks is at maximum bitrate for the calculation of rebuffering time. When the assumption does not agree with reality, caused by highly variable bandwidth and bitrate decision of MPC, Dashlet may choose an inappropriate chunk preload order. In contrast, PaCs directly sorts the expected playing time of chunks to establish the download sequence. This approach allows PaCs to avoid the problem of discrepancy between predicted chunk download completion times and actual chunk download completion times and achieves lower computational overhead than Dashlet. Moreover, chunks with high played probabilities can be downloaded first and have more chance to be preloaded at a high bitrate. However, PaCs is completely user-preference-oriented in selecting which chunks to preload. Chunks with high user preferences can be prioritized for preloading. If the predicted chunk playback order in PaCs does not match the actual playback order, chunks with higher user preferences may occupy excessive bandwidth. This can result in longer preloading times for the chunks actually played, potentially causing more rebuffering compared to Dashlet. Other preloading strategies such as work [17] and PDAS [16] try to make chunk selection and bitrate selection together, but they are hard to achieve better performance due to inappropriate chunk preload order.

Furthermore, some assumptions and implementation challenges deserve attention. Firstly, PaCs assumes that the probability distribution of user behaviors conforms to the given RP. Secondly, because the given RP is chunk-granularity, PaCs assumes that only when a chunk has been played over can the user slide the video. Thirdly, PaCs assumes that video chunks are split in 1 s time lengths, while platforms like Tiktok may split video chunks based on data size, such as 1 MB. It is necessary to align the chunk granularity and appropriately adjust the parameter Ts to adapt to different videos. Moreover, PaCs employs MPC as its default bitrate selection algorithm. This can be flexibly substituted according to specific vendor demands. Thirdly, the configuration of the recommended queue may be different among PaCs and other platforms. For instance, Tiktok sets the length of the recommended queue as 10 s, whereas in PaCs it is 5 s . Additionally, Tiktok updates the recommended queue in batches of 10 videos, while PaCs updates it each time a video is swiped away. Thus, a detailed modification of parameters should be taken into account during real-world implementation.

Finally, the user retention probability, which is derived from tremendous samples of user behavior, reflects whether the user is satisfied with the video content. Therefore, based on the user retention probability, user behavior can be simulated, i.e., the playback video would be swiped away according to the corresponding random function. In this way, the simulator can reflect the general user behavior to a certain extent. However, the user behavior depends on not only the content of the video, but also the quality of the video in reality. Thus, the simulated user behaviors would deviate from realistic ones. Therefore,

simulation with errors on retention probability, i.e., the errors of user behavior, is needed to demonstrate the robustness of PaCs.

## 4. Evaluation

### *4.1. Setup*

We implemented PaCs in the short video simulator provided for the Short Video Streaming Grand Challenge in ACM Multimedia 2022 [4]. In this simulator, users may swipe to four optional videos in the recommendation queue when the current video is playing. Moreover, the user retention probability is provided for each video chunk to determine the time of swiping. The simulator generates user behaviors using the user retention probability and repeats each simulation with a group of 50 different random seeds, resulting in 50 distinct user behaviors. Additionally, the network bandwidth is simulated based on 60 traces divided into three groups: low, medium, and high bandwidth, with each group containing 20 traces. The average and variance of the low, medium, and high bandwidth traces are 1.198 ± 0.499 MBps, 1.609 ± 0.832 MBps, and 3.576 ± 1.844 MBps, respectively. For each algorithm, we evaluate its performance under various network conditions and user behaviors using 3000 simulated environments, created from 60 network traces and 50 simulated user behaviors. Additionally, we evaluate the algorithm on a set of mixed traces, which are composed of 15 traces with highly variable bandwidth. The traces have an average bandwidth of 1.663 Mbps and a variance of 1.247. Furthermore, all videos are encoded into three different bitrates: 750 kbps, 1200 kbps, 1850 kbps, and are segmented into chunks with a fixed playing time of 1 s.

After all users finish watching the videos, the following performance metrics are collected to evaluate the effectiveness of the short video preloading schemes. The first one is the score, which is calculated according to Equation (2). This is also the standard for the Short Video Streaming Grand Challenge in ACM Multimedia 2022. By the way, $\alpha = 0.5$, $\beta = 1$, $\theta = 1$, and $\gamma = 1.85$ in Equations (1) and (2), as recommended in the simulation platform. The second parts are the terms used in Equations (1) and (2), i.e., the bitrates $Bitrate = \sum_{\epsilon} \sum_{l} b_l$, the smoothness $Smooth = \sum_{\epsilon} \sum_{l} b_{l+1} | b_l$, and the rebuffering time $Rebuf = \sum_{\epsilon} \sum_{l} r_l$ for all the video, as well as the bandwidth usage *Usage*, which refers to the total data volume of all downloaded videos. The third part is the wastage of network bandwidth *Wastage*, i.e., the data volume of downloaded video chunks that are not played.

### *4.2. Results*

#### 4.2.1. Properties of PaCs

First of all, simulations are conducted to show the basic properties of PaCs.

**Impacts of Time Scale $T_s$.** At first, the performance of PaCs with different time scales $T_s$ is evaluated under all bandwidth traces, i.e., $50 \times 60 = 3000$ simulations are conducted for each $T_s$. The cumulative distribution function (CDF) of the scores is shown in Figure 3. The curve of $T_s = 2$ is always on the left of the curves with $T_s = 3, 4, 5, 6, 7$, i.e., the scores of PaCs with $T_s = 2$ are probably smaller. To show this result in detail, we also collect all other performance indexes, normalize them with the corresponding performance index of PaCs with $T_s = 5$, and show the average and standard deviation of the normalized values in Figure 3. It indicates that the scores of PaCs with $T_s = 2$ are mainly degraded by the large *Smooth* and $Rebuf$. Specifically, PaCs with a small $T_s$ suffers from large $Rebuf$ because the future user behavior is not fully taken into account. Consequently, the mismatch between the sequence of downloading video chunks and the sequence of playing video chunks becomes significant. In other words, many video chunks may not have been downloaded before it is played, corresponding to the large $Rebuf$. Accordingly, the bitrate is switched to small, such that the rebuffering time is reduced. This is why the performance index *Smooth* is also large. In contrast, when $T_s \geq 2$, the score and QoE of PaCs is significantly improved, and the performance indexes $Rebuf$ and *Smooth* are greatly reduced. However, when $T_s$ increases to 6 and 7, the average score of PaCs decreases. This is probably because the decisions of PaCs are made based on the estimated average network bandwidth on

time scale $T_s$. When $T_s$ is large, the variation of network bandwidth would be covered and it may impact the scores of PaCs with $T_s = 6$ and $T_s = 7$. Moreover, a large $T_s$ also significantly increases the computational complexity of PaCs. PaCs with $T_s = 4$ performs well and has a smaller variance than PaCs with $T_s = 3$ in terms of performance indexes. Therefore, we set $T_s = 4$ for PaCs in the following evaluation. Dashlet employs $T_s = 5$ in simulations.

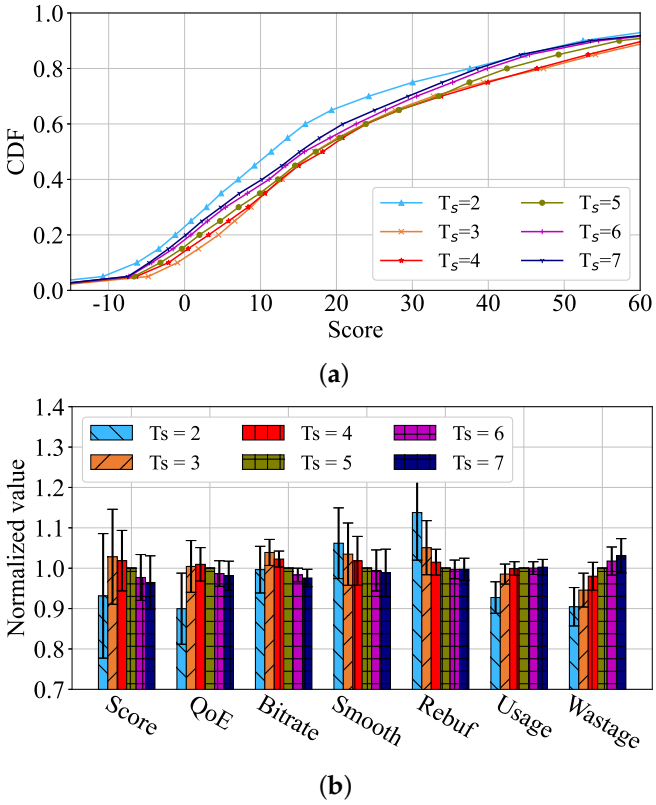


**Figure 3.** Impacts of time scale $T_s$ in PaCs. (**a**) CDF of scores. (**b**) Average performance index.

**Impacts of the pause decision.** Next, the performance of PaCs without the pause decision is also evaluated under 3000 simulations, and the performance indexes are similarly normalized based on PaCs. As shown in Figure 4, the pause decision helps PaCs reduce the bandwidth wastage and thus improve the final score. Without the pause decision, PaCs would keep on downloading video chunks, even if the video chunks may not be played. Note that PaCs without pause has a smaller *Rebuf* by downloading many video chunks. In sum, the pause decision is indispensable to improving the score of PaCs.

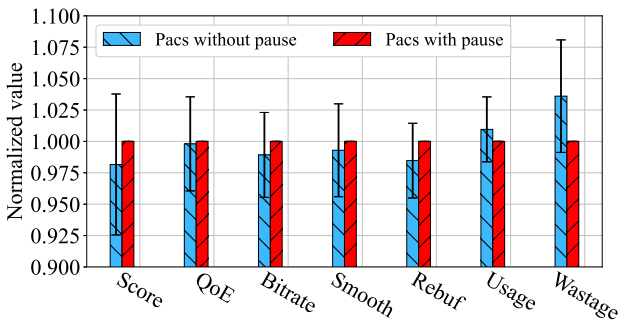


**Figure 4.** Impacts of the pause decision in PaCs.

4.2.2. Comparison with Other Schemes

The performance of PaCs is compared to existing short video preloading schemes, including the fixed-preload employed by Tiktok [1], Dashlet [18], and PDAS [16], which achieves the highest score in the Short Video Streaming Grand Challenge in ACM Multimedia 2022. We tested the average performance of different preloading strategies across 3000 simulated environments derived from 60 sets of bandwidth traces and 50 sets of user behaviors generated by the simulator using 50 different random seeds.

Figure 5 shows the CDF of scores of algorithms under all network bandwidths. PaCs achieved a maximum score under most bandwidths, especially under high bandwidths. Figure 6 shows a more specific display of the performance indexes of the preloading strategies under three network conditions. The performance indexes of other preloading strategies are normalized based on the performance indexes of PaCs.

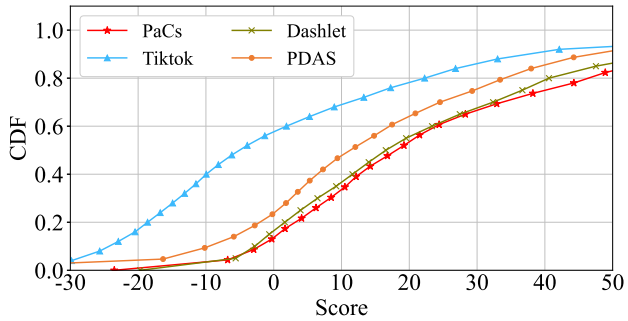


**Figure 5.** CDF of scores of algorithms under all network bandwidths.

**PaCs vs. TikTok**. TikTok achieves the lowest score in low, medium, and high bandwidth. In the high, medium, and low bandwidth, PaCs outperforms TikTok by 29%, 30%, and 40%, respectively. TikTok experiences high bandwidth waste and frequent rebuffering because it downloads a fixed number of chunks for each video in order without considering varying user behavior. As a result, downloaded video chunks are always swiped by the user without being played, while the chunks the user wants to watch are not yet downloaded.

**PaCs vs. PDAS**. PaCs has better performance compared with PDAS in low, medium, and high bandwidth, and the score of PaCs is improved by 14.6%, 9.7%, and 10.7%, respectively. Because chunks preloaded by PaCs are more in line with the sequence that chunks played, PaCs always experiences higher bitrate and lower rebuffering compared with PDAS.

**PaCs vs. Dashlet**. Compared with Dashlet, PaCs outperforms by 9% under high bandwidth, and has similar performance under medium and low bandwidth. In high-bandwidth scenarios, PaCs improves the score by 9% over Dashlet while also achieving a higher QoE and lower bandwidth wastage. Higher QoE benefits from that PaCs directly sorts chunks based on expected playing time to obtain the download sequence, while Dashlet sorts the candidate sequence based on additional rebuffering time. Moreover, Dashlet determines whether the chunks should be preloaded based on the comparison of the expected rebuffering time of the chunks with a fixed preset threshold. The threshold does not change with bandwidth; therefore, it brings Dashlet more bandwidth wastage in high-bandwidth scenarios.

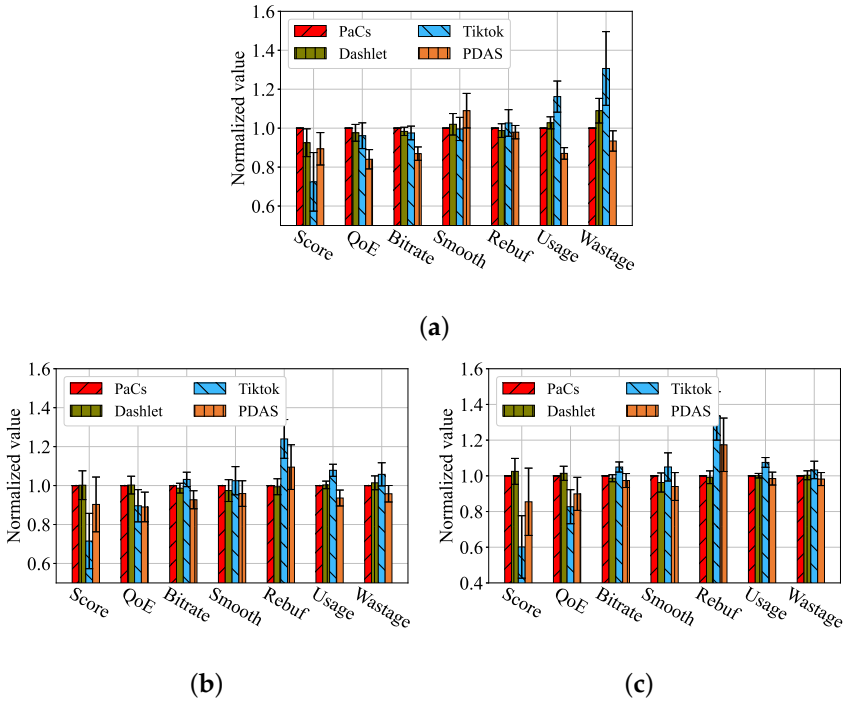


**Figure 6.** Performance comparison under different network bandwidths. (**a**) High, (**b**) medium, (**c**) low.

#### 4.2.3. Robustness of PaCs

PaCs relies on the provided RP to calculate the expected playing time of chunks and construct chunk download sequences. To assess the robustness of preloading schemes, we first compared their performance on different RPs. Next, we introduced variability in the RPs provided for the preloading schemes to examine the impact of discrepancies between the RPs and the probability distribution of tested user behavior on the performance of preloading schemes. Finally, we evaluated the effect of network prediction errors on the performance of preloading schemes.

**Impacts of different RPs.** To demonstrate the performance of preloading schemes under different kinds of RPs, we modified the RP by multiplying the played probability of each chunk by $\{1 + r\%\}$; each $\{r\%\}$ takes each value in $\{-50, -40, \ldots, 0, \ldots, 50\}$. Notably, the played probability of each chunk is limited to be smaller or equal to 1 in the modified RP. Finally, the modified RP is provided to the simulator and preloading schemes to generate tested user behaviors and construct chunk downloaded sequences, respectively. Subsequently, the performance of preloading schemes with different $r\%$ is evaluated under 3000 simulated environments, derived from 60 traces and 50 user behaviors generated by the simulator using 50 different random seeds. Next, the average scores under all simulations of preloading schemes with different $r\%$ are shown in Figure 7. The scores of all algorithms increase synchronously with the increase in $r$. Notably, PaCs and Dashlet achieved the highest scores, significantly surpassing PADS and TikTok algorithms in all cases. The larger the value of $r$, the later the video sliding time of user behavior is generated by the modified RP. PaCs can preload chunks in time, and the probability of preloaded chunks being wasted is low. On the contrary, the smaller the value of $r$, the earlier the video sliding time of user behavior generated by the modified RP. PaCs frequently fails to preload chunks in time, and numerous preloaded chunks are wasted, resulting in high rebuffering and bandwidth wastage.

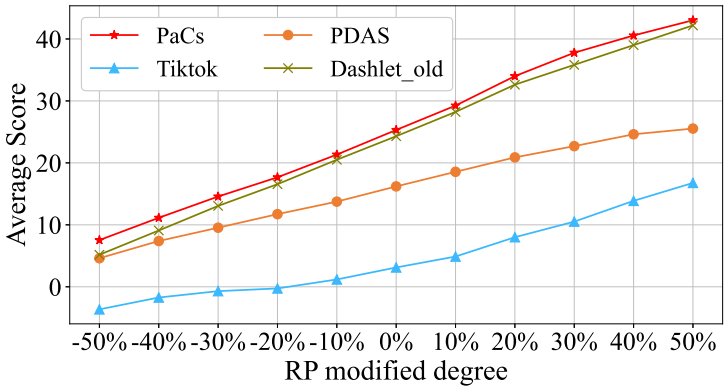


**Figure 7.** Impacts of RP.

**Impacts of RP discrepancy.** There may be a discrepancy between the RP provided for the preloading schemes and the probability distribution of tested user behaviors because the RP is only based on the video chunk playing times of numerous users, which may not represent all users. To demonstrate the performance of preloading schemes when discrepancy exists between the provided RP and the probability distribution of user behavior, we modified the RP provided for preloading schemes by multiplying the played probability of each chunk in RP by $\{1 + d\%\}$, and $d$ takes each value in $\{-50, -40, \ldots, 0, \ldots, 50\}$. We defined the RP with added discrepancy $d\%$ as the modified RP and the RP without added discrepancy $d\%$ as the unmodified RP. Notably, the played probability of each chunk is limited to be smaller or equal to 1 in the modified RP. Subsequently, the experiment also evaluated 3000 simulated environments composed of 50 user behaviors and 60 traces. The difference is that the modified RP is used for preloading schemes, while the simulator uses the unmodified RP to generate user behavior. Next, the average scores under all simulations of preloading schemes with different $d\%$ are shown in Figure 8.

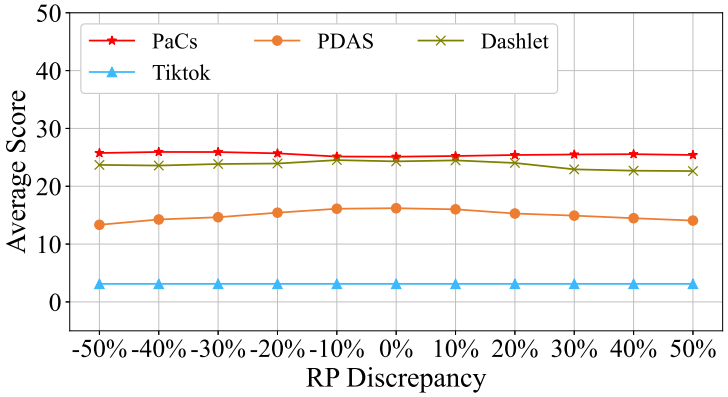


**Figure 8.** Impacts of RP discrepancy.

The graph illustrates how the performance of the preloading schemes varies with different discrepancies in the RP provided for preloading schemes. As shown in Figure 8, PaCs achieves the best performance, closely followed by Dashlet. In fact, the differences among the two algorithms are minimal. Specifically, when $d < 0$, PaCs's score still remains above 90% of PaCs's Score without discrepancy. Even with $d = \pm 50\%$, PaCs's score maintains at least 90% of PaCs's score without discrepancy. PaCs has a high tolerance to RP discrepancy because no significant changes occur in chunk downloaded sequences when the provided RP involved discrepancy. When the provided RP involved discrepancy, the expected time of all candidate chunks synchronously changes, increasing or decreasing. Accordingly, the downloaded chunk sequences change a little, and PaCs maintains good performance.

**Impact of network prediction errors.** We evaluated the performance of preloading schemes in the presence of network prediction errors. Specifically, when calculating the download time of chunks in the process of MPC, the future bandwidth is directly obatined from the bandwidth traces instead of being obtained through prediction. Then, the obained future bandwidth is multiplied by $\{1 + e\%\}$. $e$ represents network prediction error and takes each value in $\{-50, -40, \ldots, 0, \ldots, 50\}$. Subsequently, the performance of preloading schemes with different $e\%$ values is evaluated under all simulations, i.e., $50 \times 60 = 3000$ simulations are conducted for each $e$. Next, the average scores under all simulations of preloading schemes with different $e\%$ are shown in Figure 9.

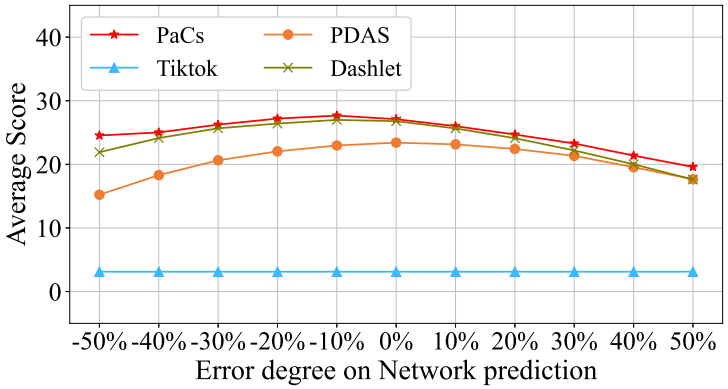


**Figure 9.** Impacts of network prediction errors.

As shown in Figure 9, the PaCs score shows a slightly downward trend when $e < 0$ and $e > 0$, as $e$ decreases and increases, respectively. Even with $e\% = -50\%$ and $e\% = 50\%$ on future bandwidth prediction, the PaCs score can maintain an 80% score without network prediction errors. This is because PaCs relies solely on the expected playing time of chunks for sorting, while bandwidth fluctuations only impact the final bitrate selection. Therefore, when $e < 0$, the decrease in PaCs score mainly results from too low of a bitrate selection. The bandwidth is not fully utilized. When $e > 0$, the decrease in PaCs score mainly results from too high of a bitrate selection. Chunks can not be preloaded in time, and rebuffering occurs. Overall, PaCs has a high tolerance on network prediction errors.

**Impact of highly variable network conditions.** To verify the performance of the algorithm under highly variable network conditions, we evaluated the performance of different preloading strategies across 750 simulated environments created from 15 mixed network traces and 50 simulated user behaviors. The average bandwidth of mixed traces is 1.663 MBps. The corresponding variance is 1.247 MBps, which is large enough to represent the highly variable network conditions. Moreover, aside from videos provided by [4], different types of videos from Tiktok are employed. Additionally, the corresponding user retention probability of each video is generated randomly. Figure 10 shows the CDF of scores of different algorithms under highly variable network conditions. Compared to Dashlet, PADS, and TikTok, PaCs achieved the highest score. Moreover, the detailed performance indexes are shown in Figure 11. PaCs outperforms TikTok, PADS, and Dashlet in terms of score by 29.9%, 13.2%, and 7.2%, respectively. PaCs achieves the highest QoE, the highest bitrate, the lowest rebuffering, and the best smoothness. Unlike Dashlet, PaCs directly sorts the chunks based on the expected playback time to obtain the preloaded chunk order. This approach allows PaCs to avoid the problem of discrepancy between predicted chunk download completion times and actual chunk download completion times.

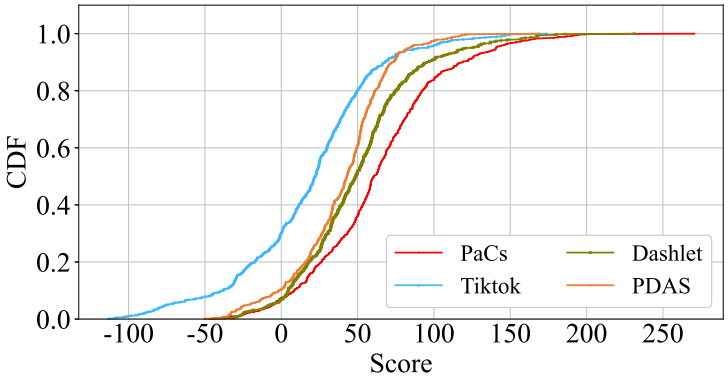


**Figure 10.** CDF of scores of algorithms under highly variable network conditions.

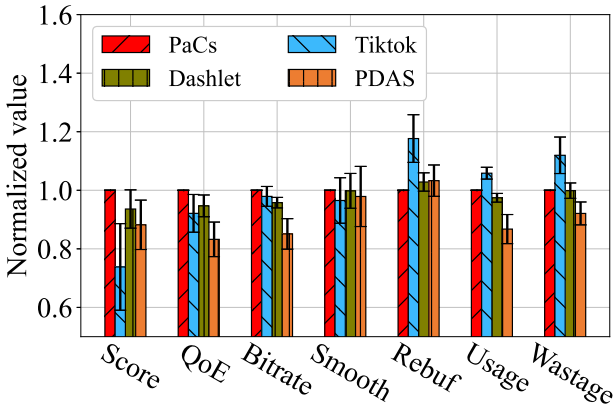


**Figure 11.** Performance comparison under highly variable network conditions.

## 5. Conclusions

In this paper, we reveal that existing short video preloading schemes, such as Dashlet and PDAS, suffer from inappropriate chunk preload order and large computation complexity due to highly variable network conditions and user behaviors. To address this issue, we propose PaCs, which fully considers the impact of uncertain user behavior and decouples the selection of chunk preload order from bitrate selection, thus reducing the computation complexity and achieving better performance. Specifically, PaCs calculates the expected playing time of video chunks and downloads the video chunk with the smallest expected playing time at first. After the selection of the video chunk, PaCs employs the classic MPC algorithm to select the bitrate, which can be flexibly substituted according to specific vendor demands. Furthermore, PaCs would decide whether to pause or execute the downloading process, i.e., delay the downloading of video chunks without affecting QoE, thus reducing bandwidth wastage due to the video being swiped away. Extensive simulation results show that PaCs achieves better performance than both Dashlet and PDAS across various network bandwidths, user behaviors, and video types. Specifically, in the scenario with highly variable network conditions, PaCs outperforms TikTok, PADS, and Dashlet in terms of score by 29.9%, 13.2%, and 7.2%, respectively.

**Author Contributions:** Conceptualization, S.F., Z.Y. and S.T.; data curation, S.F., G.Y. and Y.S.; funding acquisition, Z.Y., S.T. and W.J.; methodology, S.F., G.Y. and S.T.; project administration, Z.Y.; resources, S.T.; software , S.F. and G.Y.; supervision, Z.Y., S.T. and W.J.; validation, Y.S.; visualization, G.Y. and Y.S.; writing—original draft, S.F.; writing—review and editing, Z.Y., S.T. and W.J. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research was partly funded by the Science and Technology Innovation Program of Hunan Province, China under grant number 2023RC3047, and the Education Bureau of Hunan Province, China under grant number 202401001745.

**Data Availability Statement:** Data are contained within the article.

**Conflicts of Interest:** Author Sen Fu was employed by the company China Aerospace Science and Industry Corporation. The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

## References

1. Chris Marcus. Short Form Video Statistics and 2023 Marketing Trends. Available online: https://www.colormatics.com/article/short-form-video-statistics-and-2020-marketing-trends/ (accessed on 20 September 2024).
2. Statista. Number of Global Downloads of ByteDance's Major Apps as of August 5, 2024. Available online: https://www.statista.com/statistics/1343066/bytedance-most-downloaded-apps/ (accessed on 5 August 2024).
3. Statista. Most Popular Social Networks Worldwide as of April 2024, by Number of Monthly Active Users. Available online: https://www.statista.com/statistics/272014/global-social-networks-ranked-by-number-of-users/ (accessed on 27 April 2024).
4. Zuo, X.; Li, Y.; Xu, M.; Ooi, W.T.; Liu, J.; Jiang, J.; Zhang, X.; Zheng, K.; Cui, Y. Bandwidth-Efficient Multi-video Prefetching for Short Video Streaming. In Proceedings of the 30th ACM International Conference on Multimedia, Chengdu, China, 20–24 October 2021.
5. Qian, S.-Z.; Xie, Y.; Pan, Z.; Zhang, Y.; Lin, T. DAM: Deep Reinforcement Learning Based Preload Algorithm with Action Masking for Short Video Streaming. In Proceedings of the 30th ACM International Conference on Multimedia, Lisbon, Portugal, 10–14 October, 2022; pp. 7030–7034.
6. Yin, X.; Jindal, A.; Sekar, V.; Sinopoli, B. A Control-Theoretic Approach for Dynamic Adaptive Video Streaming over HTTP. In Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication, London, UK, 17–21 August 2015; pp. 325–338.
7. Zhang, X.; Ou, Y.; Sen, S.; Jiang, J. SENSEI: Aligning Video Streaming Quality with Dynamic User Sensitivity. In Proceedings of the 18th USENIX Symposium on Networked Systems Design and Implementation (NSDI), Virtual, 12–14 April 2021; pp. 303–320.
8. Yan, F.Y.; Ayers, H.; Zhu, C.; Fouladi, S.; Hong, J.; Zhang, K.; Levis, P.; Winstein, K. Learning in Situ: A Randomized Experiment in Video Streaming. In Proceedings of the 17th Usenix Conference on Networked Systems Design and Implementation, Santa Clara, CA, USA, 25–27 February 2020; pp. 495–512.
9. Mao, H.; Netravali, R.; Alizadeh, M. Neural Adaptive Video Streaming with Pensieve. In Proceedings of the Conference of the ACM Special Interest Group on Data Communication, Los Angeles, CA, USA, 21–25 August 2017; pp. 197–210.
10. Sun, Y.; Yin, X.; Jiang, J.; Sekar, V.; Lin, F.; Wang, N.; Liu, T.; Sinopoli, B. CS2P: Improving video bitrate selection and adaptation with data-driven throughput prediction. In Proceedings of the 2016 ACM SIGCOMM Conference, Florianópolis, Brazil, 22–26 August 2016; pp. 272–285.
11. Huang, T.; Zhou, C.; Zhang, Ru.; Wu, C.; Yao, X.; Sun, L. Comyco: Quality-aware adaptive video streaming via imitation learning. In Proceedings of the 27th ACM International Conference on Multimedia, Nice, France, 21–25 October 2019; pp. 429–437.
12. Kan, N.; Jiang, Y.; Li, C.; Dai, W.; Zou, J.; Xiong, H. Improving generalization for neural adaptive video streaming via meta reinforcement learning. In Proceedings of the 30th ACM International Conference on Multimedia, Lisbon, Portugal, 10–14 October 2022; pp. 3006–3016.
13. Akhtar, Z.; Nam, Y.S.; Govindan, R.; Rao, S.G.; Chen, J.; Katz-Bassett, E.; Ribeiro, B.; Zhan, J.; Zhang, H. Oboe: Auto-tuning video ABR algorithms to network condition. In Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication, Budapest, Hungary, 20–25 August 2018; pp. 44–58.
14. Zhang, H.; Zhou, A.; Lu, J.; Ma, R.; Hu, Y.; Li, C.; Zhang, X.; Ma, H.; Chen, X. OnRL: Improving mobile video telephony via online reinforcement learning. In Proceedings of the 26th Annual International Conference on Mobile Computing and Networking, London, UK, 21–25 September 2020; pp. 1–14.
15. Phong, N.T.; Huong, T.T.; Nam, P.N.; Thang, T.C.; Nguyen, D. Joint preloading and bitrate adaptation for short video streaming. *IEEE Access* **2023**, *11*, 121064–121076. [CrossRef]
16. Zhou, C.; Ban, Y.; Zhao, Y.; Guo, L.; Yu, B. PDAS: Probability-Driven Adaptive Streaming for Short Video. In Proceedings of the 30th ACM International Conference on Multimedia, Lisbon, Portugal, 10–14 October 2022; pp. 7021–7025.
17. Wu, X.; Zhang, L.; Cui, L. QoE-Aware Download Control and Bitrate Adaptation for Short Video Streaming. In Proceedings of the 30th ACM International Conference on Multimedia, Lisbon, Portugal, 10–14 October 2022; pp. 7115–7119
18. Li, Z.; Xie, Y.; Netravali, R.; Jamieson, K. Dashlet: Taming Swipe Uncertainty for Robust Short Video Streaming. In Proceedings of the 20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23), Boston, MA, USA, 17–19 April 2023; pp. 1583–1599.
19. Zhang, H.; Ban, Y.; Zhang, X.; Guo, Z.; Xu, Z.; Meng, S.; Li, J.; Wang, Y. APL: Adaptive Preloading of Short Video with Lyapunov Optimization. In Proceedings of the 2020 IEEE International Conference on Visual Communications and Image Processing (VCIP), Macau, China, 1–4 December 2020; pp. 13–16.
20. Zhang, G.; Zhang, J.; Liu, K.; Guo, J.; Lee, J.Y.B.; Hu, H.; Aggarwal, V. DUASVS: A Mobile Data Saving Strategy in Short-Form Video Streaming. *IEEE Trans. Serv. Comput.* **2023**, *16*, 1066–1078. [CrossRef]

21. He, J.; Hu, M.; Zhou, Y.; Wu, D. Liveclip: Towards intelligent mobile short-form video streaming with deep reinforcement learning. In Proceedings of the NOSSDAV 2020—Proceedings of the 2020 Workshop on Network and Operating System Support for Digital Audio and Video, Part of MMSys 2020, Istanbul, Turkey, 10–11 June 2020; pp. 54–59.
22. Li, Y.; Zheng, Q.; Zhang, Z.; Chen, H.; Ma, Z. Improving ABR Performance for Short Video Streaming Using Multi-Agent Reinforcement Learning with Expert Guidance. In Proceedings of the NOSSDAV '23: Proceedings of the 33rd Workshop on Network and Operating System Support for Digital Audio and Video, Vancouver, BC, Canada, 10 June 2023; pp. 58–64.
23. Li, J.; Xu, H.; Ma, M.; Yan, H.; Xue, C.J. Alfie: Neural-reinforced adaptive prefetching for short videos. In Proceedings of the 2022 IEEE International Conference on Multimedia and Expo (ICME), Taipei, Taiwan, 18–22 July 2022; pp. 1–6.
24. Sun, F.; Yang, H.; Lin, T.; Zhang, Y.; Chen, Z.; Chen, Z.; Yan, J. CLAPS: Curriculum Learning-Based Adaptive Bitrate and Preloading for Short Video Streaming. In Proceedings of the 2023 IEEE 25th International Workshop on Multimedia Signal Processing (MMSP), Poitiers, France, 27–29 September 2023; pp. 1–6.
25. Xie, Y.; Zhang, Y.; Lin, T.; Pan, Z.; Qian, S.-Z.; Jiang, B.; Yan, J. Short video preloading via domain knowledge assisted deep reinforcement learning. *Digit. Commun. Netw.* 2024, *in press*.
26. Dai, B.; Li, H.; Wang, Y. aCroSS: Ai-driven cross-layer adaptive streaming for short video applications. *Comput. Netw.* **2024**, *254*, 110832. [CrossRef]
27. Guo, J.; Zhang, G. A Video-Quality Driven Strategy in Short Video Streaming. In Proceedings of the 24th International ACM Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems, Alicante, Spain, 22–26 November 2021; pp. 221–228.


**Disclaimer/Publisher's Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.