ALFIE: NEURAL-REINFORCED ADAPTIVE PREFETCHING FOR SHORT VIDEOS

Jingzong Li[*], *Hong Xu*[†], *Ming Ma*[‡], *Haopeng Yan*[‡] *and Chun Jason Xue*[*]

[*]City University of Hong Kong, [†]The Chinese University of Hong Kong, [‡]Kuaishou
[*]jingzong.li@my.cityu.edu.hk, [†]hongxu@cuhk.edu.hk
[‡]{maming, yanhaopeng03}@kuaishou.com, [*]jasonxue@cityu.edu.hk

ABSTRACT

Short videos have received extraordinary success in recent years. To provide smooth playback and avoid rebuffering delay, prefetching upcoming videos is commonly used in cellular networks. Current prefetching designs fall short in dealing with bandwidth overhead, especially the exit overhead of downloaded but unconsumed chunks due to user exit. Measurement from a large short video platform shows that exit overhead accounts for up to 43.5% of bandwidth overhead. Thus we build Alfie, a bandwidth-efficient short video prefetching algorithm via reinforcement learning. Essentially Alfie adjusts prefetching based upon user viewing patterns in addition to network conditions. We demonstrate that Alfie outperforms the state of the art by up to 26.8% in overall performance while reducing the exit overhead by up to 84.9%.

***Index Terms*—** Short video streaming, Prefetching, Bandwidth overhead, Deep reinforcement learning

1. INTRODUCTION

Short video consumption has skyrocketed in recent years driven by apps such as TikTok (Douyin in China), Kuaishou, and Twitter. The number of short video users has reached 873 million in China, representing 88.3% of its total netizens [1]. To improve quality of experience (QoE) for users, prefetching has been widely adopted by large short video companies [2]. By downloading part of the upcoming videos in the recommendation queue in advance, prefetching greatly reduces the startup delay (i.e. the lag between user swiping and video playback) which is particularly important for short videos that typically last no more than 20 seconds. It also reduces the rebuffering time during playback which occurs when the buffer becomes empty due to user's random swiping and changing network conditions [3].

Despite the benefits, aggressive prefetching incurs *bandwidth overhead* when the downloaded chunks are not eventually consumed if the user swipes to the next video or simply exits the session. This results in increased bandwidth cost which is already one of the major components of the operation costs for the video platform [4], and also higher cellular data usage at the user's side. To better understand the bandwidth overhead of prefetching, we carry out an empirical study using a real-wold trace collected from Kuaishou [5] with over 400 million view sessions. We find that, simple policies used in production that always prefetch a fixed number of videos and the first few chunks from each video lead to over 2700TB of downloaded but unwatched content on a daily basis. As mentioned, this is partly due to user swiping to the next video, which some work has recently started to investigate [2]. In addition, we find that bandwidth overhead due to user exiting the app, or *exit overhead* in short, also takes up to 43.5% of the total and has not been studied to our knowledge. As we look closely into the culprit of exit overhead, we identify that user watching behavior exhibits a long tail pattern with nearly 52% sessions consuming no more than six videos. This implies that prefetching just a few videos could already incur salient bandwidth overhead relative to the total bandwidth consumption of the session.

Motivated by these observations, we argue that prefetching should adapt to the user's specific viewing patterns as well as the changing network conditions. Temporally, as a user watches more videos, prefetching for her should be more conservative to reduce the exit overhead as the likelihood of departure is increasing. Then spatially, if a user's viewing behavior is typical based on historical data, prefetching should be more conservative as she usually just watches a few videos before leaving. If a user is in the long tail, however, prefetching can be more aggressive to further improve QoE without too much exit overhead. Finally, prefetching also needs to adapt to the network conditions as cellular networks often suffer from frequent and drastic bandwidth fluctuations [3]. If the network condition is likely to deteriorate, one may choose to prefetch more to cope with the bandwidth shortage.

The central question we seek to address in the following is: how to design such an adaptive prefetching policy? This is challenging because essentially, both user behaviors and network conditions are almost impossible to model precisely. User behaviors are intrinsically hard to capture as the decision to swipe (to the next video) or leave the session is influenced by many factors including user's interest towards the content,

This work is supported in part by funding from the Research Grants Council of Hong Kong (11209520) and a gift fund from Microsoft (CUHK grant no. 6906276).

her time available, perceived viewing quality, and even her state of mind [6]. Cellular network condition is also affected by the surrounding environment, user mobility, and even interference from the neighboring cells, among others.

To tackle this challenge, we resort to deep reinforcement learning (DRL) [7] which has witnessed success in solving networking problems, including video streaming [8]. Prefetching is intrinsically a sequential and far-sighted process which considers long-term accumulative reward (whether a prefetched chunk will be watched or not is unknown at decision time); such a setting perfectly fits into DRL. Our DRL model exploits a set of features about user behavior (e.g. historical session duration, viewing time of past videos), network condition (e.g. recent network bandwidth), and upcoming videos, as state of the environment, and learns a good policy that optimizes the long-term benefit with QoE gain and bandwidth overhead. To make this work, we design a novel reward mechanism for bandwidth overhead, apply domain knowledge to avoid meaningless exploration, and develop a short video streaming simulator to train Alfie.

We make the following contributions in this work:

- We empirically demonstrate the bandwidth overhead of static prefetching for short videos by using a large-scale production trace from a real-world short video platform.
- Based on our findings, we propose Alfie, a novel adaptive prefetching algorithm for short videos based on DRL.
- We implement and evaluate Alfie using production traces. We show the effectiveness and generalization capability of Alfie with an average improvement of 22.2% compared to state of the art.

2. MOTIVATION

We start by motivating adaptive prefetching with an empirical analysis of static prefetching using a real-world trace.

In current practice, prefetching commonly follows a simple static policy which always downloads the first i videos from the top of the recommendation queue, and for each video always downloads the first j chunks. We denote this family of policies as S-i-j. Here we consider two instances of this static policy, S-3-3 and S-5-6. Note downloading is performed sequentially in prefetching; downloading of the current video takes priority and is controlled by the player independent of prefetching.

We collect a production trace of over 400 million sessions of short video viewing for a 24-hour period starting on March 1st, 2021 from Kuaishou [5], one of the largest short video platforms in China. The trace contains detailed information about the videos (length, chunk size, etc.), users (time on each video, session duration), and the network (average download throughput of each video). To analyze the bandwidth overhead, we build a high-fidelity simulator which is also needed for DRL training as will be explained in §3.3. We replay the entire trace in our simulator together with the static prefetching policies, and obtain the bandwidth overhead as the total bandwidth consumption minus the total size of consumed video chunks. By using the session duration information, we also obtain the exit overhead as all the prefetched but unwatched data due to exit.

Scheme	Daily Bandwidth Overhead (TB)	Daily Exit Overhead (TB) & Ratio	Annual Bandwidth Overhead Cost Range ($1M USD)	Annual Exit Overhead Cost Range ($1M USD)
S-5-6	2795	1216 (43.5%)	[∼41, ∼122]	[∼18, ∼53]
S-3-3	1738	365 (21.0%)	[∼25, ∼76]	[∼5, ∼16]

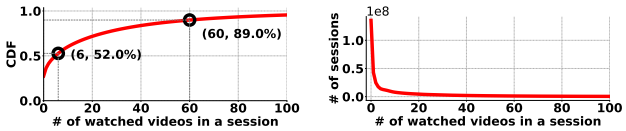
Table 1: Bandwidth overhead of two static prefetching methods using our trace. The bandwidth cost is estimated using the minimum and maximum prices from data in [9].

Key Finding 1*: Static prefetching results in significant bandwidth overhead, including exit overhead.*

Table 1 shows the bandwidth overhead stands at 2795TB and 1738TB for S-5-6 and S-3-3, respectively. This translates to tens of millions dollars of cost annually. In particular, exit overhead amounts to 43.5% and 21.0% of the total, respectively, for the two methods. The limited prior work on short videos [2, 10] considers swiping overhead which occurs when the user swipes to the next video without consuming all the prefetched data, but has not discussed exit overhead which is also important as we have just shown.

To study the cause of exit overhead, we perform another analysis on user's viewing behavior. Note that a session is defined between a user entering and leaving the app, in which at least one video is viewed. Fig. 1 shows the distribution of number of videos viewed in each session in our one-day trace.

Key Finding 2*: User watching behavior is long-tailed.*

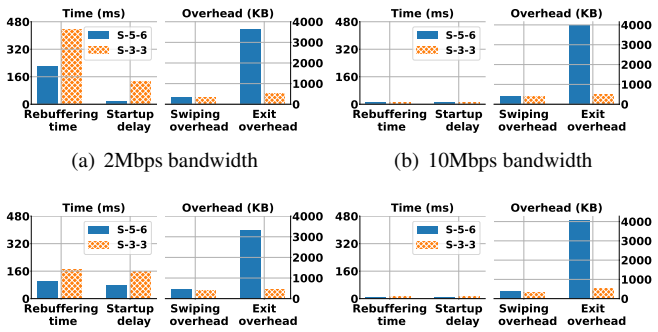


(a) CDF of number of short videos viewed in each session. (b) Histogram of number of videos viewed in each session.

Fig. 1: Long-tailed user viewing behavior summarized across all sessions in our trace.

We observe from Fig. 1(a) that the majority (89%) of the sessions consume no more than 60 videos. Further, nearly 52% of sessions consume no more than six short videos, meaning a large number of users leave after watching just a few videos. From Fig. 1(b) we validate that the session distribution based on the number of videos viewed follows a long-tailed power-law distribution with an exponent $\alpha = 2.18$. We argue that the long-tailed user behavior is a major cause of exit overhead, simply because it implies that even prefetching just a few videos could lead to non-negligible bandwidth overhead. In the extreme case for the 29% of the sessions that consume only one video, any prefetched data is abandoned.

The above two findings motivate us to develop bandwidth-efficient prefetching policies that minimize the exit overhead in particular and the bandwidth overhead in general.



(a) 2Mbps bandwidth (b) 10Mbps bandwidth

(c) Fast swiping (d) Slow swiping

Fig. 2: QoE and overheads of two static methods in different environments. For simplicity we consider constant bandwidth networks here. Users take 4.9s and 45s on average to swipe to the next video in fast and slow swiping settings.

***Key Finding 3**: Static policies do not adapt well.*

A simple idea to reduce the bandwidth overhead is to prefetch less. The challenge is in striking a balance between the overhead and QoE gain: when can we prefetch less without sacrificing the QoE, and how much should we prefetch exactly? Intuitively we need a dynamic policy here that adapts to the changing environment in answering the above challenge, and in this section we show that static policies indeed do not work well in different environments.

We consider different environments reflected by network bandwidth (2Mbps and 10Mbps) and user swiping behavior (45s and 4.9s for average swiping time). Fig. 2 illustrates the corresponding QoE and bandwidth overhead obtained using the simulator with our trace. Observe that when the network bandwidth is low and/or when user tends to swipe quickly, S-5-6 is better as it offers less rebuffering time and startup delay. Yet in other scenarios, the more conservative S-3-3 delivers a better tradeoff as it yields less overheads with very similar QoE. There is no one-size-fits-all solution here, and it is necessary to dynamically adjust the prefetching strategy.

Related work. Chen et al. [11] carried out a general study in the characteristics of Douyin videos, and Lu et al. [12] looked into the user motivations behind using Douyin from a HCI perspective. They lack a concrete system solution to improve short video streaming. Recently, LiveClip [2] and DUASVS [10] adopt learning-based methods for short video prefetching. Yet, they do not consider exit overhead.

3. DESIGN

We now present the detailed design of Alfie that consists of a RL module (§3.1), a slow start mechanism (§3.2) and a short video streaming simulator (§3.3), as shown in Fig. 3.

3.1. DRL Setup and Algorithm

We start by explaining how DRL is used here to model the prefetching problem.

Why DRL? DRL fits well in the prefetching problem for two main reasons: (1) Long-term optimization. Prefetching is intrinsically a sequential and far-sighted process which considers long-term accumulative rewards. In contrast, supervised and unsupervised learning are usually one-shot and myopic considering instant rewards only [7]. (2) Generalization and robustness. Static schemes inherently suffer from generalization problem as described in §2 and the optimal solution needs to be re-computed whenever the environment changes. DRL adapts to network conditions and user behaviors by learning from a large corpus of real-world traces and actual interactions [8]. Its output policy is robust in handling different environments.

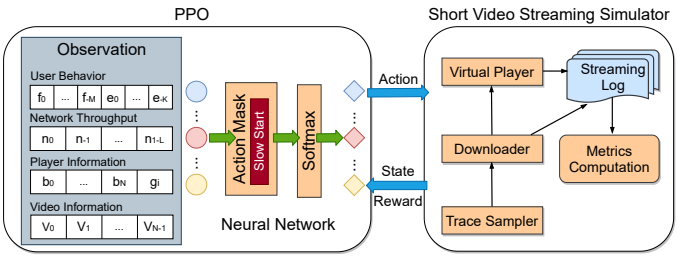


Fig. 3: Overview of Alfie's architecture.

We describe how the state and action spaces are defined in Alfie. Like typical RL, in each step when the system finishes prefetching one chunk of video, the agent receives rewards from the environment, observes the current state, and takes a prefetching action.

State Space. In Alfie, at a given step i the state space $S_i = [U_i, W_i, P_i, V_i]$ comprises of the following:

1. *User behavior*. User swiping affects the prefetching range and aggressiveness. We use F_i to represent the time spent on each video for the past M videos, i.e. $F_i = [f_0, f_{-1}, \ldots, f_{-M}]$ where f_0 is the time spent on the current video so far. Besides swiping behavior, we also mine user's exit behavior denoted as E_i. More specifically, $E_i = [e_0, e_{-1}, ..., e_{-K}]$ describe the number of videos viewed in the latest K sessions where e_0 is the number of videos viewed in the current session. Overall, user behavior features are represented as $U_i = [F_i, E_i]$.
2. *Network throughput*. Here $W_i = [n_0, \ldots, n_{1-L}]$ denotes the average download throughput of the latest L chunks that finish downloading. Note to reflect the network condition timely, the chunks here include both those of the current video and those of prefetched videos sorted by their download completion times.
3. *Playback information*. The sizes of downloaded data for the current video plus all N upcoming videos from the recommendation queue are represented as $B_i = [b_0, \ldots, b_N]$, where b_0 is downloaded size of the current video, b_1 is the size of prefetched data of the first video at the top of the queue, and so on. Besides, playback information also includes the play progress of the current video g_i, thus $P_i = [B_i, g_i]$.
4. *Chunk information*. We also consider the size of candidate chunks of all N upcoming videos, $V_i = [v_0, \ldots, v_{N-1}]$.

Action Space. The action space is represented by a scalar A_i which takes integer value from 0 to N. When $A_i = 0$, it means Alfie chooses not to prefetch anything. Otherwise, Alfie prefetches the next chunk not in the buffer for the A_i-th video from the recommendation queue (each video may have different prefetching progress). Here we simplify the design by assuming that prefetching always follows the playback order in downloading the chunks, and each action only downloads one chunk. This makes sense since the short videos are commonly played sequentially. Whenever Alfie finds that the first upcoming video has not been prefetched at all, it always chooses to prefetch it in this step, i.e. $A_i = 1$, which is clearly the best action in terms of reducing the likelihood of rebuffering. This is enforced through action masking [13] as detailed in §4.1.

Reward Function. The reward function is crucial for reinforcement learning. There are two issues that complicate the reward definition: (1) QoE and bandwidth overhead are two contrary goals when defining the reward; (2) more importantly, the overhead signal is irregularly delayed because we do not instantly know whether the downloading chunk will be consumed or not until the video is swiped away midway or completely consumed.

Our reward function $R(S_i, A_i, S_{i+1})$ is shaped from two aspects as shown below:

$$R(S_i, A_i, S_{i+1}) = \begin{cases} R_{\text{idle}}(S_i, S_{i+1}), & \text{if } A_i = 0, \\ R_{\text{prefetch}}(S_i, A_i, S_{i+1}), & \text{otherwise.} \end{cases} \tag{1}$$

Here A_i is the action taken at step i, R_{idle} stands for the reward if current action is idle, while R_{prefetch} is the reward for prefetching a chunk.

First, when the RL agent chooses to idle, the reward is:

$$R_{\text{idle}}(S_i, S_{i+1}) = \begin{cases} -1, & \text{if } T_{\text{idle}} > 0, \\ \alpha, & \alpha > 0, \text{otherwise,} \end{cases} \tag{2}$$

where T_{idle} denotes the rebuffering time of the current video during the idling period from S_i to S_{i+1}. Rebuffering occurs when the next chunk to be played is not downloaded by the playback time, and each time it happens we record its duration. Note that the goal of RL agent is to maximize the cumulative discounted reward, even though sometimes the punishment for early actions are inflicted on the current action, the delayed feedback can still be captured by the cumulative reward [7]. Here $\alpha > 0$ is a positive reward coefficient which controls the trade-off between QoE and bandwidth overhead. If α is large, the agent tends to idle more often because it can still get rewards even if rebuffering events frequently occur; if α is close to 0, the agent prefetches more aggressively.

Second, prefetching the next chunk from video A_i in the recommendation queue leads to $R_{\text{prefetch}}(S_i, A_i, S_{i+1})$ as determined by Algorithm 1. Rebuffering should be avoided and thus punished as the top priority in line 3. Punishing rebuffering also implicitly minimizes the startup delay. Then to consider overhead, we need to know whether a chunk will be played or not. In Alfie's current design we assume this information is obtained from session traces collected offline. Line 1 gets the number of remaining videos to be played in the current session, and line 5 punishes downloading a video that will not be viewed due to user exit. Then in line 7 the time user spent on A_i is obtained from the trace, and we punish the swiping overhead when the timestamp of the chunk j is beyond the user's lingering time in line 9. Finally, positive reward is given to prefetching chunk that is eventually consumed by the user. Intuitively the earlier in a video the chunk is, the higher the reward needs to be. Thus, we design the reward based on its relative position as shown in line 11.

Algorithm 1 Calculation of reward R_{prefetch}

Input: A_i: video selected for prefetching; j: position of the chunk to be prefetched in A_i; $T(S_i, A_i, S_{i+1})$: rebuffering time during downloading chunk j

1: $h \leftarrow$ GetVideoNum(VideoId, SessionTrace) ▷ Get the number of remaining videos from the session trace; VideoId is the current video's ID
2: **if** $T(S_i, A_i, S_{i+1}) > 0$ **then**
3: $\quad R_{\text{prefetch}} \leftarrow -1$ ▷ punishment for rebuffering
4: **else if** $A_i > h$ **then**
5: $\quad R_{\text{prefetch}} \leftarrow -1$ ▷ punishment for prefetching a video that will not be watched due to user exit
6: **else**
7: $\quad s \leftarrow$ GetStayingTime(A_i, SessionTrace) ▷ Get the time user spent on video A_i from the trace
8: $\quad$ **if** $j > s$ **then**
9: $\quad\quad R_{\text{prefetch}} \leftarrow -1$ ▷ punishment for downloading a chunk that will not be viewed due to user swiping
10: $\quad$ **else**
11: $\quad\quad R_{\text{prefetch}} \leftarrow \beta * (s - j)/s$
12: **return** R_{prefetch}

3.2. Slow Start

We also propose a *slow start* mechanism to better adapt to the long-tailed user viewing behavior, drawing inspiration from TCP's slow start. Prefetching range is restricted by a window which is progressively expanded. Initially at the beginning of a viewing session, the window size is only one which means Alfie only prefetches one video. It grows by one with one more consumed video until it reaches N, the recommendation queue size. Slow start is incorporated into Alfie's DRL model by action masking.

3.3. Simulator

It is impractical to directly train Alfie in the real world simply because a single short video lasts dozens of seconds. Therefore Alfie relies on a discrete-event based short video streaming simulator, similar to many prior arts [3, 8]. Simulators in these previous systems only support video-on-demand in a sequential loop. As shown in Fig. 3, Alfie's simulator works as follows: the *trace sampler* loads session traces and network traces, and injects all user swiping and exit events into an event queue. During streaming, the *downloader* executes

downloading events using network throughput received from the trace sampler and actions from the RL model. Note this includes both downloading current video and prefetching upcoming videos. The *virtual player* controls the playback and accounts for rebuffering and swiping events. Performance metrics such as rebuffering time and overhead are recorded into the log. Our simulator can simulate 244 hours of short video streaming in only 8 minutes.

4. IMPLEMENTATION AND EVALUATION

4.1. Evaluation Setup

Implementation Alfie is implemented by using one of the leading RL libraries, RLlib in Ray [14], and a toolkit for developing RL environments, OpenAI Gym [15]. We adopt PPO [16] as our default RL training algorithm; the discount factor $\gamma \in (0, 1)$ is set to 0.99, which implies the agent cares a lot about rewards in the distant future relative to the immediate ones. Our RL model is composed of two fully-connected layers with 256 units each, and the tanh activation function. Action masking is done by setting the corresponding output probability to 0. The reward coefficients α and β are set to 0.1 and 1 according to our experience. The number of past videos K, past sessions M, and throughput measurements L are all set to 8. The recommendation queue size N is 5 and each chunk is 1-second long following industry practice. Our prototype consists of ∼2.5K LoC in python.

Dataset. Our dataset includes network traces and session traces. Following [8], we use 80% of the traces as the training set while the remaining 20% as the testing set. (1) Network traces. We collect a corpus of proprietary real-world network traces (*Kuaishou trace*) covering various network conditions (0.2-23.8 Mbps, 4G/LTE). Each data point in the trace represents 1-second throughput. Besides, we also adopt the HSDPA trace [17] (*Public trace*), collected from mobile networks in Norway. (2) Session traces. We take a subset of our complete production traces from Kuaishou as introduced in §2. Specifically, the trace used in evaluation contains details of randomly sampled ∼10000 real-world sessions with ∼1 million short video viewings. Note all short videos are encoded by VBR with variable bitrate.

Performance Metrics. We use four performance metrics: two for QoE and two for bandwidth overhead. (i) *Rebuffering time* T, (ii) *Startup delay* D: the lag between the user swiping event and the time for playback to begin; (iii) *Swiping overhead* W_s, and (iv) *Exit overhead* W_e, which are already defined and discussed in detail in §1 and §2. Unless otherwise noted, the four metrics are calculated by averaging across all sessions. Note that unlike in VoD, video bitrate does not need to be considered in QoE for short videos because their bitrate is determined when the recommendation queue is generated.

Taken everything above, we consider the following *negative utility* by combining all the above four metrics:

$$U = T + D + 0.1 \times W_s + 0.1 \times W_e \quad (3)$$

A smaller U is better as the metrics all reflect negative effects.

Network	Scheme	Rebuffering time (ms)	Startup delay (ms)	Swiping overhead (KB)	Exit overhead (KB)	Negative Utility
Kuaishou trace	Oracle	249	111	7	0	365
	Next-One	444	252	2116	21149	17616
	S-3-3	594	141	194	475	1222
	S-5-6	464	114	224	3873	3559
	S-5-12	435	118	343	7228	6061
	LiveClip	488	129	1383	2965	3779
	Alfie	318	121	247	543	1014
Public trace	Oracle	119	75	4	0	197
	Next-One	232	170	2353	26078	21080
	S-3-3	303	90	221	506	922
	S-5-6	232	81	244	3993	3395
	S-5-12	217	81	371	7586	6085
	LiveClip	246	86	1342	3146	3596
	Alfie	156	83	271	407	733

Table 2: Overall performance.

Here T and D have the same weight since startup delay is also crucial in short video streaming. All terms are in ms in U.

Baseline Schemes. We compare Alfie with the state-of-the-art prefetching methods. **I. Oracle**: this policy downloads chunks sequentially in the ideal case when future information from user is known (i.e. user swiping and exit times). **II. Next-One** [2]: this is adopted by Douyin. It always downloads the current video completely, and then starts to download the next video in full. It stays idle then until a swiping event occurs. **III. Static**: this is introduced in §2. In addition to S-3-3 and S-5-6, we consider S-5-12 in this section. **IV. LiveClip** [2]: a learning-based prefetching algorithm for short video streaming. It selects chunk from the current video and next two upcoming videos to download.

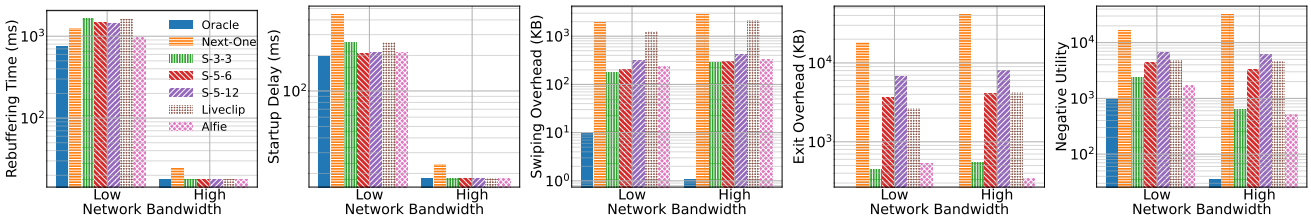
4.2. Overall Performance

Table 2 summarizes the overall performance across all test sessions. We make the following observations: (1) Alfie outperforms existing methods by large margins mostly. It has the lowest negative utility except Oracle, and the improvement is up to 20.5% compared to the best non-clairvoyant method (S-3-3). (2) Alfie reduces swiping overhead and exit overhead without impairing QoE. In particular, the exit overhead is reduced by up to 84.9% compared to LiveClip. In fact Alfie's rebuffering time is even better than aggressive prefetching such as S-5-12. The reason is that Alfie does not rigidly prefetch one video after another; instead, it adaptively fills the buffer in a flexible manner to ensure fast startup by considering user behavior.

4.3. Generalization

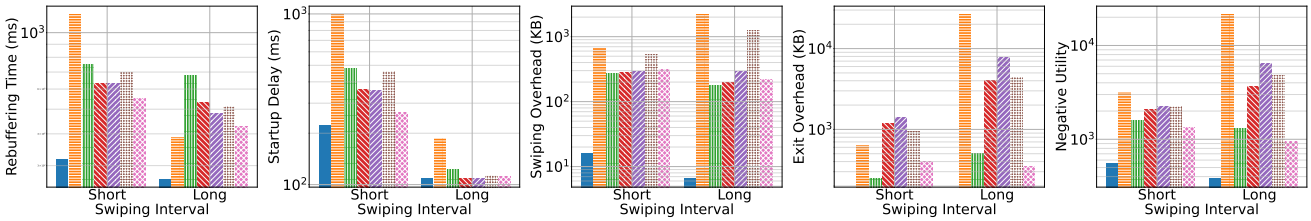
Alfie can encounter vastly different environments in terms of network conditions and user swiping patterns. To evaluate the generalization ability of Alfie, we consider four scenarios and select particular traces as our test data, and use the same prefetching policy learned through training in §4.1.

We select two network traces that represent low (avg. 1.48Mbps) and high (avg. 20Mbps) bandwidth scenarios, respectively. The same session trace is used in both scenarios for fairness. Fig. 4 shows Alfie delivers 18.9%–26.8% improvement in overall utility over existing methods. The



(a) rebuffering Time (ms) (b) Startup Delay (ms) (c) Swiping Overhead (KB) (d) Exit Overhead (KB) (e) Negative Utility

Fig. 4: Performance under different network conditions. Note the log scale of the y-axis.



(a) Rebuffering Time (ms) (b) Startup Delay (ms) (c) Swiping Overhead (KB) (d) Exit Overhead (KB) (e) Negative Utility

Fig. 5: Performance under different behavior patterns. Note the log scale of the y-axis.

improvement comes from two aspects: 1) Alfie provides a bandwidth-efficient prefetching policy which effectively reduces bandwidth overhead especially exit overhead. 2) Alfie can incorporates throughput history information into its state space to optimize for near future network characteristics.

Conversely, we select two session traces that represent short (avg. 4.9s) and long (avg. 45s) swiping intervals of users, and use the same network trace for both scenarios. Fig. 5 shows that Alfie achieves 13.3% improvement in rebuffering time compared to best static policy S-5-12 when user performs fast swiping; Next-One, S-3-3, and LiveClip perform poorly in QoE metrics, because they all prefetch in a fixed short range that does not handle fast swiping events well. Another reason for LiveClip's poor performance is that it assumes constant bitrate (CBR) encoding, which makes it unable to accommodate varying chunk sizes. When the swiping interval is long, we can see Alfie is slightly worse than Next-One in rebuffering time as Next-One prefetches aggressively at the expense of bandwidth.

4.4. Analysis of Learned Policy

To understand the insights learned by Alfie, we further analyze its prefetching behaviors according to its learned policy. We have the following findings: (1) Alfie tends to prefetch more aggressively in lower bandwidth networks to avoid rebuffering, and prefetch conservatively when bandwidth is high. (2) With fast swiping, Alfie's policy prefers the first chunk of each video as soon as possible before later contents. (3) It tends to become more conservative when the session is about to end. We leave as future work a more extensive study of Alfie's learned policies, as well as whether those insights can be translated into simpler prefetching mechanisms.

5. CONCLUSION

To our best knowledge, this is the first work that thoroughly investigates the bandwidth overhead including exit overhead of short video streaming, and proposes a neural network based bandwidth-efficient prefetching design. Alfie adapts to variable and unseen environments by learning from massive past experiences, and delivers better overall performance.

6. REFERENCES

[1] "CNNIC.," https://www.cnnic.com.cn/IDR/.

[2] Jianchao He, et al., "LiveClip: Towards Intelligent Mobile Short Video Streaming with Deep Reinforcement Learning," in *Proc. ACM NOSSDAV*, 2020.

[3] Xiaoqi Yin, et al., "A Control-theoretic Approach for Dynamic Adaptive Video Streaming over HTTP," in *Proc. ACM SIGCOMM*, 2015.

[4] "Interim Report 2021.," https://ir.kuaishou.com/corporate-filings/financial-information.

[5] "Kuaishou Technology.," https://www.kuaishou.com/.

[6] Pierre Lebreton, et al., "Study on User Quitting in the Puffer Live TV Video Streaming Service," in *International Conference on Quality of Multimedia Experience (QoMEX)*, 2021.

[7] Richard S. Sutton, et al., *Reinforcement learning: An introduction*, MIT press, 2018.

[8] Hongzi Mao, et al., "Neural Adaptive Video Streaming with Pensieve," in *Proc. ACM SIGCOMM*, 2017.

[9] "CDN Pricing.," https://cdn.reviews/cdn-pricing-comparison/.

[10] Guanghui Zhang, et al., "DUASVS: A Mobile Data Saving Strategy in Short-form Video Streaming," *IEEE Transactions on Services Computing*, pp. 1–1, Feb. 2022.

[11] Zhuang Chen, et al., "A study on the characteristics of douyin short videos and implications for edge caching," in *Proceedings of the ACM Turing Celebration Conference-China*, 2019.

[12] Xing Lu, et al., "Fifteen Seconds of Fame: A Qualitative Study of Douyin, a Short Video Sharing Mobile Application in China," in *International Conference on human-computer interaction*, 2019.

[13] Shengyi Huang, et al., "A Closer Look at Invalid Action Masking in Policy Gradient Algorithms," *arXiv:2006.14171*, 2020.

[14] "Rllib.," https://docs.ray.io/en/releases-1.5.0/rllib.html.

[15] "OpenAI Gym.," https://gym.openai.com/.

[16] John Schulman, et al., "Proximal Policy Optimization Algorithms," *arXiv:1707.06347*, 2017.

[17] "HSDPA.," https://datasets.simula.no/hsdpa-tcp-logs/.