JumpDASH: LLM-Based Content Perception for Intelligent Jumping DASH in Mobile Adaptive Video Streaming

Hanling Wang, Tianli Zhou, Qing Li, *Senior Member, IEEE*, Yong Jiang, *Member, IEEE*, and Gabriel-Miro Muntean, *Fellow, IEEE*

***Abstract*—Traditional Adaptive Bitrate (ABR) schemes assume that users watch videos sequentially, focusing solely on the sequential downloading of video chunks. However, these schemes often result in significant degradation of Quality of Experience (QoE) when users skip directly to their preferred segments. To address this issue, we propose JumpDASH, which leverages Large Language Model (LLM)-based content perception to enhance mobile adaptive video streaming. First, JumpDASH incorporates a low-cost video text summarization module based on large language models, enabling users to identify and navigate to the most relevant sections of videos. Second, we introduce a dynamic partitioned buffer and a Proximal Policy Optimization (PPO)-based ABR algorithm to facilitate prefetching video chunks corresponding to the perceived points of interest, along with differentiated encoding techniques to further minimize rebuffering. Extensive experiments conducted using real trace datasets under actual network conditions show that JumpDASH improves QoE by 13.82% to 262.94% compared to existing ABR technologies.**

***Index Terms*—Mobile video streaming, large language model, jumping prefetch, video content perception.**

I. Introduction

ACCORDING to recent reports [1], [2], video streaming accounts for about 82% of the total Internet traffic, with Video on Demand (**VoD**) representing a significant share. Unfortunately, the highly dynamic nature of mobile networks, poses significant challenges when providing consistent high-quality video transmission to users. To address these challenges, Dynamic Adaptive Streaming over HTTP (**DASH**) [3] and Adaptive Bitrate (**ABR**) algorithms [4], [5], [6], [7], [8], [9], [10] were proposed to adjust video encoding and delivery rates in response to fluctuating network conditions.

Since the advent of DASH, researchers have shown that different segments of a video often attract varying levels of user attention and interest [11], [12], [13], [14], [15]. For instance, in movies, ensuring smooth playback of a high-action scene can have a more significant impact on overall user Quality of Experience (**QoE**) levels than on movie closing credits. Despite this, most existing ABRs (*e.g.,* Pensieve [4], MPC [6], BOLA [5], Fugu [7], and Merina [16]) treat all video segments uniformly, focusing on optimizing overall video quality through adaptive bitrate adjustments without accounting for content differences between video chunks.

To date, some *content-aware* ABRs [11], [13], [17], [18] classify video chunks into *high priority (hp)* chunks and *low priority (lp)* chunks, where *hp* chunks are those that offer higher perceptual quality gain or carry more semantic information. By allocating bandwidth to download higher-bitrate versions of *hp* chunks and lower-bitrate versions of *lp* chunks, these schemes have been shown to be effective for good quality video playback. However, most popular on-demand video platforms (*e.g.,* YouTube [19], Netflix [20], Bilibili [21]) allow users to skip ahead using the progress bar to access more interesting video parts. When users skip video content, the priority schemes encounter issues such as buffer underflow and prolonged rebuffering for *hp* chunks, as they rely on the buffer to handle network throughput fluctuations.

To address these issues, it is essential to design a mechanism capable of differentiating video content, enabling users to quickly locate segments of interest within a video. Existing methods for video content understanding typically leverage perceptual quality [11], [13], [18] or semantic information [12], [14], [15], [22], [23], [24], [25]. Perceptual quality measures how the human eye perceives video quality degradation during transmission, but does not aid in understanding video content. Semantic information reflects the level of interest in the events shown in video frames, but current methods [12], [14], [15], [22], [25] largely rely on labor-intensive processes or lightweight deep learning models, both of which have limitations. On one hand, labor-intensive techniques [12], [14], [22] typically require humans to watch each video (*e.g.,* crowdsourcing) to understand the content, making them prohibitively expensive and impractical for large-scale deployment. On the

Received 24 December 2024; revised 25 July 2025; accepted 14 September 2025; approved by IEEE TRANSACTIONS ON NETWORKING Editor H. Hassanieh. This work was supported in part by the Project of the Department of Strategic and Advanced Interdisciplinary Research of the Pengcheng Laboratory under Grant 2025QYA001, in part by the National Key Research and Development Program of China under Grant 2022YFB3105000, and in part by Shenzhen Key Laboratory of Software Defined Networking under Grant ZDSYS20140509172959989. *(Corresponding author: Qing Li.)*

Hanling Wang, Tianli Zhou, and Yong Jiang are with the Pengcheng Laboratory, Shenzhen, Guangdong 518055, China, and also with Tsinghua Shenzhen International Graduate School, Tsinghua University, Shenzhen, Guangdong 518055, China (e-mail: wanghl03@pcl.ac.cn; ztl21@mails.tsinghua.edu.cn; jiangy@sz.tsinghua.edu.cn).

Qing Li is with the Pengcheng Laboratory, Shenzhen, Guangdong 518055, China (e-mail: liq@pcl.ac.cn).

Gabriel-Miro Muntean is with the Performance Engineering Laboratory, Dublin City University, Dublin, D09 V209 Ireland (e-mail: gabriel.muntean@dcu.ie).

Digital Object Identifier 10.1109/TON.2025.3611495

other hand, lightweight deep learning models [23], [24] often struggle to effectively interpret User Generated Videos (**UGV**), which generally encompass a wide variety of video types.

In this paper, we propose *JumpDASH*, an innovative solution to leverage Large Language Model (**LLM**)-based content perception to enhance mobile video streaming quality. JumpDASH introduces two key innovations: low-cost video content perception based on Multimodal LLMs (**MLLMs**) and jump prefetching to efficiently load key content when users skip to interesting segments.

First, performing a straightforward full-frame analysis using MLLMs for video content perception is prohibitively expensive. To address this challenge, we propose a low-cost method to generate text summaries from videos, which features three key components: a Textual Content Agent (**TCA**), a Visual Content Agent (**VCA**), and a Video Segment Summary Module (**VSSM**). TCA leverages subtitle gap and coherence analysis to identify video segments that can be understood solely through subtitles. For segments requiring visual analysis, VCA employs video encoding data to select critical frames from a local perspective and utilizes deep learning to globally filter out unnecessary frames based on content and temporal diversity. This eliminates redundant analysis, ensuring that only the most relevant frames are processed by MLLMs. VSSM constructs a Chain of Thought (**CoT**) using LLMs to combine subtitle data and visual text descriptions into a cohesive and comprehensive text summary of the entire video.

Second, to support high quality video transmission, unlike traditional methods, JumpDASH employs a non-sequential prefetching strategy (denoted "jump prefetching"). For high-quality *hp* chunks, JumpDASH prioritizes content significance over strict sequential retrieval. Additionally, it mitigates potential increases in rebuffering times caused by jump prefetching while reducing image quality fluctuations in *hp* chunks. This is achieved through fine-grained encoding and efficient bitrate allocation, ensuring a seamless viewing experience.

To evaluate JumpDASH, we implemented a prototype featuring a *Nginx*-based content server and a *ffplay*-based prefetch client. Comprehensive experimental results show that JumpDASH significantly improves overall QoE, achieving enhancements ranging from 13.82% to 262.94%, outperforming the latest ABR technologies by a large margin. This evaluation highlights that JumpDASH is a robust solution for enhancing video streaming experience across diverse and dynamic network conditions. Our contributions are:

- We design a novel video streaming solution, JumpDASH, which seamlessly integrates a video text summarization module with optimized video transmission strategies to improve QoE during non-sequential video viewing.
- We introduce a low-cost video text summarization module that efficiently derives content insights from subtitles and video frames, enabling users to locate their desired video segments accurately and effortlessly.
- We propose adaptive streaming enhancement modules that facilitate the prefetching of important video segments with minimal disruption to ongoing playback.

II. Related Works

A. Assessing Video Content

User experience when interacting with videos is influenced by various factors. However, the major user experience influencing factors can be related to user perception of quality and content semantic information.

Perceptual quality refers to human-perceived video quality. In early works [26], visual quality assessment (**VQA**) was conducted through objective evaluation metrics such as Peak Signal-to-Noise Ratio (**PSNR**) and the Structural Similarity Index Measure (**SSIM**), but they did not fully capture the human video experience. Subjective video quality evaluation in terms of metrics, such as the Mean Opinion Score (**MOS**), offers a numerical measure of the human-judged overall quality of an event or experience and is both more appropriate and more accurate, but it is time consuming and complex. Recently, some machine learning-based VQA techniques that combine both subjective and objective metrics have been proposed. For example, Netflix introduced the Video Multi-Method Assessment Fusion (**VMAF**), a video quality metric which uses a reference video and outperformed all other metrics on all recent compression standards [11], while ITU-T proposed P1204.3, a no-reference video quality metric very useful for real-time video quality estimation [27]. However, the drawback of perceptual quality assessment is that it does not understand the video content and it cannot be used to identify the key elements within the video.

Semantic information refers to the specific data within the video frames that users find interesting. To identify semantic information within videos, three primary approaches are used: (*i*) *Crowdsourcing-based solutions* [12], [14] involve engaging a large number of human workers to watch videos and manually select preferred segments to identify key moments; (*ii*) *Highlight detection-based methods* [23], [24] leverage lightweight DNN models to automatically detect interesting video segments; (*iii*) *User retention-based approaches* [15], [25] determine the most engaging frames by analyzing viewer retention rates for each frame. Crowdsourcing and user retention-based methods require extensive viewership to determine the semantic information of videos, limiting their applicability to only a small subset of videos. For example, Sensei [22] conducted separate crowdsourcing experiments for each video to assess users' quality sensitivity by collecting quality ratings on multiple degraded renderings, which is both time-consuming and costly. Similarly, highlight detection-based approaches require training multiple deep learning models to identify specific moments, making them impractical for extracting semantic information from large number of UGVs.

In addition, a series of studies [28], [29], [30] have explored the use of LLMs/VLMs for video content understanding, typically focusing on novel model designs or relying solely on frame-based inputs to filter content and reduce computational cost. In contrast, our work leverages existing LLMs and emphasizes the combined use of subtitles and selective frame sampling to enhance efficiency.

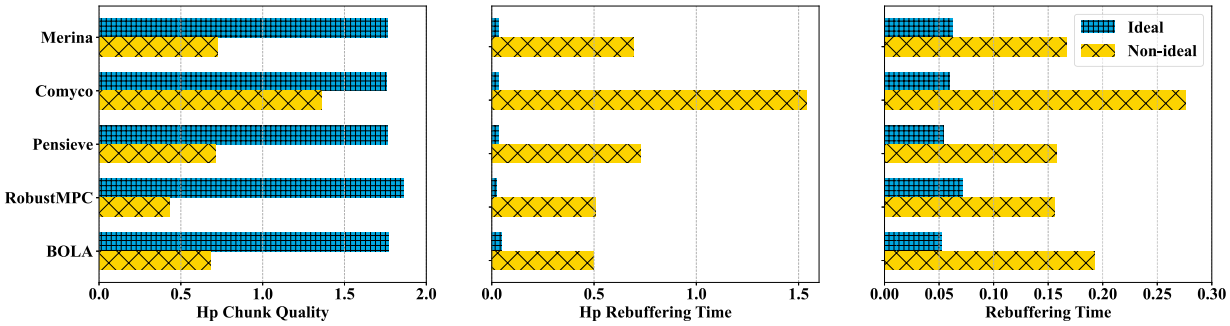


Fig. 1. Comparison of ABR performance between the ideal scenario, where videos are viewed sequentially, and the non-ideal scenario, where users jump to segments of interest.

B. Content-Aware ABR

An important goal for research is to design solutions to improve the perceived quality of the videos delivered across various networks and ABR approaches were among the most successful. Some ABR schemes (*e.g.,* Pensieve [4], MPC [6], BOLA [5], and Merina [16]) treat all video chunks equally without considering content differences when assigning adaptive bitrates. However, user QoE is inherently influenced by video (and scene) content [13], [22]. For instance, scenes with rapid motion and higher user engagement benefit more from higher bitrates compared to less critical video segments.

Considering this observation, some ABR schemes assign different bitrates to video chunks based on their estimated importance. For example, HotDASH [17] prioritizes the download of users' preferred video chunks. It employs three deep reinforcement learning (**DRL**) models to make prefetching decisions and select the bitrate for video chunks. Comyco [18] determines chunk bitrates by feeding video chunk size and VMAF into a model with expert policies. It also accounts for network condition drift and periodically updates the DRL model using selected data. DAVS [13] explicitly considers video content differences by categorizing video chunks into dynamic chunks (*i.e.,* scenes with dynamic and complex content) and static chunks, assigning higher bitrates to the dynamic chunks for enhanced QoE. These solutions are very good for classic video consumption, but are very limited in their usefulness in context where there is non-sequential user viewing pattern (*e.g.,* the users are allowed to jump and view any video segments of interest, regardless of their location within the video stream).

III. Research Motivation

A. Challenges of Non-Sequential Video Viewing

Existing ABR solutions typically assume that users watch videos sequentially, and therefore, the video chunks are downloaded sequentially. However, most video platforms (*e.g.,* YouTube, Netflix, and Bilibili) provide users with the ability to skip to specific video segments by sliding the progress bar. If a user jumps to play a *hp* chunk, existing ABRs often suffer from prolonged video rebuffering or are forced to significantly reduce the bitrate of the video chunk. This leads to a substantial degradation in QoE levels for users.

To demonstrate this phenomenon, we conducted a real trace-driven test using widely adopted video encoding settings [4], [16], [18] in ABR scenarios. The test utilized a dataset of 20 videos collected from YouTube, including genres such as games, news, sports, and documentaries. We designate the first K video chunks at the beginning of each key segment as *hp* chunks, as in Section § V-D, because these regions are where users are more likely to seek or jump to. For each video, we randomly selected five jump combinations from the set of *hp* chunks to represent user behaviors at various playback times. These jump targets simulate scenarios in which users drag the progress bar to reach their desired segments. We evaluated five state-of-the-art ABRs: (*i*) MPC [6], (*ii*) BOLA [5], (*iii*) Pensieve [4], (*iv*) Comyco [18], and (*v*) Merina [16].

To examine whether existing ABRs can maintain high QoE in non-sequential viewing scenarios, we divided the experiment into two groups. The first group represents the ideal scenario, where users watch videos sequentially. The second group simulates realistic non-ideal user behavior, where users jump to specific segments (*i.e., hp* chunks) while viewing. Figure 1 presents the results, where we evaluated three key metrics: the average quality of *hp* video chunks, the average *hp* rebuffering time (defined as the rebuffering time specifically when viewing *hp* chunks), and the average rebuffering time across all video chunks.

In the non-ideal scenario, where users jump to segments of interest, the average quality of *hp* chunks decreases significantly by approximately 22.67%-76.92%. Additionally, the average *hp* rebuffering time increases dramatically by 904.37%-4812.37%, while the overall rebuffering time rises noticeably by 117.44%-366.94%. This increase in rebuffering time further exacerbates the decline in QoE under real-world conditions. These findings highlight that existing ABRs are unsuitable for real-world scenarios involving potential user "jumps".

Given these observations, we argue it is crucial to rethink traditional ABR strategies to account for the dynamic viewing habits of users. The inability of current ABR systems to handle non-sequential viewing patterns highlights the need for a more flexible and user-centric approach. To address this gap, this paper introduces a novel solution aimed at transforming how video streaming services adapt to user behavior, with a particular emphasis on improving the viewing experience in scenarios involving jumps.

B. Large Language Models for Content Perception

Currently, the video transmission domain lacks an effective method for understanding diverse UGV content from a semantic perspective. Fortunately, with the release of ChatGPT by OpenAI [31], LLMs based on the Transformer architecture have achieved remarkable success. Further advancements in these models have led to the development of MLLMs [32], [33], [34], [35], which have also shown strong performance across various image and video types. These models highlight the potential for a deeper semantic understanding of diverse video content. However, directly applying MLLMs for video understanding incurs significant computational costs. For example, using OpenAI's public GPT4-V API [36] to analyze a 1-hour video at 512×512 resolution frame by frame, the cost of input tokens alone[1] can reach 220.32.

In this paper, we aim to leverage the powerful capabilities of MLLMs for video content perception while mitigating the associated high computational costs. To achieve this, we propose a novel method that combines subtitle information analysis with selective frame sampling, enabling the generation of video text summaries at a much lower cost.

IV. JumpDASH Overview

A. JumpDASH Workflow

Figure 2 illustrates JumpDASH's system architecture. It consists of a *Content Server* and a *Prefetch Client*, which exchange metadata and video chunks during the streaming session. The JumpDASH solution's major steps are indicated in the figure and are briefly introduced next.

- *Step 1*: The raw video is processed by a *Low-Cost Video Text Summary Module* to generate a text summary and determine the sequence numbers of *hp* chunks;
- *Steps 2-3*: The *Differentiated Encoding* Module retrieves the sequence numbers of *hp* chunks. It first encodes the entire video at a fixed bitrate and then encodes the *hp* chunks into finer-grained sub-chunks;
- *Step 4*: The video text summary is delivered to users, allowing them to quickly grasp key points of the video;
- *Steps 5-8*: The ABR mechanism collects data on the video streaming status and determines which video chunks to download. The downloaded chunks are stored in the *Dynamic Buffer* while the user watches the video;
- *Step 9*: Users can navigate the video using the progress bar, selecting segments of interest based on the provided video text summary.

B. Content Server

The *Content Server* utilizes LLMs to enable efficient comprehension of video content. By extracting textual summaries and identifying key points from subtitles and selected video frames, it performs intelligent segmentation and summarization of video content. This process integrates a Textual Content Agent (TCA) to analyze subtitles using the Subtitle Gap Index and coherence analysis, alongside a Visual Content Agent (VCA) to evaluate selected critical frames. Together, these agents generate video text summaries that enable users to navigate quickly to segments of interest while significantly reducing computational overhead.

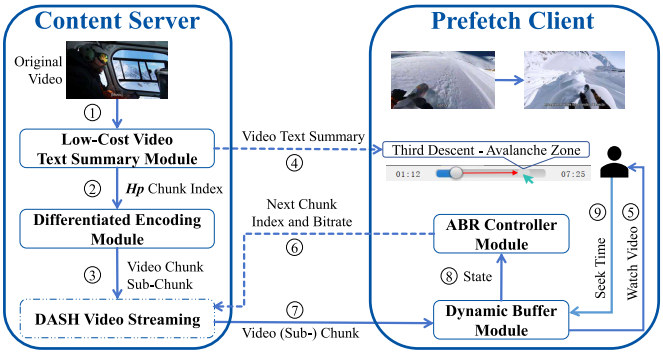


Fig. 2. The workflow of JumpDASH.

The *Content Server* processes video requests through an HTTP service, encoding videos at multiple bitrates to create various representations. These representations are segmented into chunks and classified as *hp* chunks or *lp* chunks based on semantic content analysis. To further enhance the viewing experience, the server employs a *Differentiated Encoding* module to minimize quality fluctuations and reduce rebuffering time. Specifically, it subdivides *hp* chunks into smaller sub-chunks using SSIM-based sub-chunk boundary selection and assigns different bitrates to the sub-chunks through VMAF-guided optimization with simulated annealing. This ensures smoother streaming performance, particularly within critical video segments.

Additionally, the *Content Server* compiles all relevant video information (*e.g.,* chunk size, duration, quality) and *hp* chunk descriptions (start times and fine-grained durations) into a Media Presentation Description (**MPD**) file, facilitating seamless adaptive video delivery and playback using DASH.

C. Prefetch Client

The *Dynamic Buffer* manages chunk buffering by dynamically adjusting buffer sizes for sequential and non-sequential prefetching based on current network conditions and buffer status. This approach ensures efficient bandwidth utilization, with a focus on prioritizing the download of future *hp* chunks.

The *ABR Controller* employs a reinforcement learning algorithm to determine whether to download the next video chunk in sequence or prefetch a non-sequential *hp* chunk. This decision-making process optimizes QoE by considering factors such as anticipated network throughput, client cache status, and the contextual significance of the video content.

V. Low-Cost Video Text Summary

Low-cost video text summary was introduced to analyze video content efficiently. By identifying the *hp* chunks in the video, it ensures that the most valuable segments for users are prioritized and streamed at high quality in real-time according to the network conditions.

[1]Typically, the computational cost for LLMs or MLLMs is calculated based on the total number of input and output tokens.

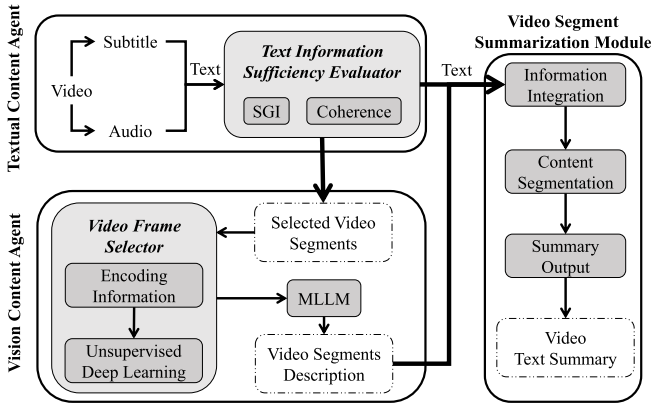


Fig. 3. Low-cost video text summary.

The workflow of low-cost video text summary is illustrated in Figure 3. It comprises three key components: (*i*) TCA, which extracts and evaluates critical textual information from subtitles or audio to identify segments understandable through text alone; (*ii*) VCA, which uses advanced frame filtering to detect and analyze visually significant frames, effectively determining *hp* chunks requiring visual interpretation; and (*iii*) VSSM, which combines insights from TCA and VCA to segment the video into coherent parts and generate concise summaries for each segment, providing an organized and comprehensive content overview.

A. Textual Content Analysis

TCA extracts key textual information from subtitles and audio to identify video segments that do not require additional visual context for comprehension. Initially, TCA checks whether the input video contains subtitle information. If subtitles are unavailable, speech recognition is performed using Whisper [37] to generate subtitles with timestamps. This transcription process provides a textual representation of the video content, preserving key temporal information synchronized with the audio-visual experience, forming the foundation for detailed content analysis. Based on the generated subtitle files, TCA performs a thorough analysis to pinpoint video segments that can be analyzed with textual information alone.

We employ a novel Text Information Sufficiency Evaluator comprising two key components: SGI (Subtitle Gap Index) evaluation and textual coherence judgment. SGI quantifies the time intervals between subtitles, assessing how much of the content can be comprehended using subtitles alone. Textual coherence judgment further assesses the completeness and clarity of the subtitle content.

SGI Evaluation. To assess whether video content can be fully understood using subtitles and audio alone, we introduced a metric called Subtitle Gap Index (**SGI**). SGI is defined as the average gap (in seconds) of adjacent subtitles in a video. It quantifies the continuity of subtitles in a video by measuring the time intervals between subtitles. A higher SGI value indicates larger gaps between subtitles, potentially hindering viewers' ability to comprehend the video content solely through text. If the calculated SGI exceeds a specified threshold θ, it suggests that the subtitle information may be insufficient for a complete understanding of the content, requiring visual support. Conversely, an SGI below this threshold indicates that the subtitles alone provide adequate information for comprehensive video understanding.

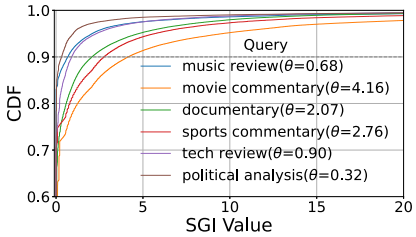


Fig. 4. The CDF of SGI.

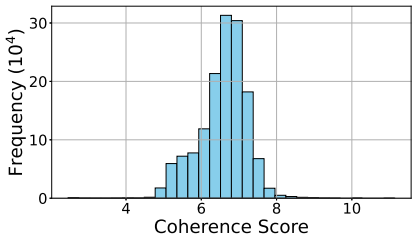


Fig. 5. Subtitle coherence distribution.

Setting threshold θ**.** Videos of explanatory types, such as sports commentary, movies, TV series narrations, and documentaries, are often meticulously prepared, enabling viewers to understand the content well through text alone. To determine a suitable threshold θ, we collected subtitles from 1,784 YouTube videos of these types, totaling 1,588 hours, and calculated their SGI values. Figure 4 illustrates the Cumulative Distribution Function (**CDF**) of SGI values across all videos. The SGI values at the 90th percentile range from 0.32 to 4.16 across different categories. To accommodate the diversity of subtitles in videos, we set the threshold θ at 4.16. This means that if the SGI exceeds 4.16, the video content is deemed incomprehensible through text alone, necessitating visual analysis.

Subtitle Coherence Judgment. Subtitle coherence plays a critical role in audience comprehension. To evaluate subtitle coherence, we employ a pretrained GPT-2 model [38] for analysis. GPT-2 assigns a coherence score to each subtitle segment by processing the subtitle text and using the loss of the output text as the coherence score. This loss reflects the narrative fluidity and logical consistency of each segment, with lower scores indicating greater clarity and more likely to be understood independently.

By analyzing subtitles from the same 1,784 YouTube videos already mentioned, we generated a histogram of coherence scores, shown in Figure 5. Most videos exhibit coherence scores concentrated around 7, indicating consistent coherence across different categories of explanatory videos. Additionally, Figure 6 illustrates the CDF of coherence scores, revealing that over 90% of subtitle scores fall below a coherence score of 7.27. Based on this analysis, we set the coherence score threshold at 7.27, ensuring that the majority of video content

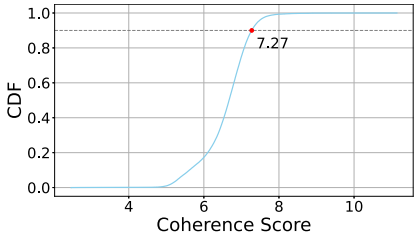


Fig. 6. CDF of subtitle coherence scores.

can be comprehended through subtitles alone without requiring additional visual information.

In summary, SGI values capture the temporal gaps between subtitles, while coherence scores quantitatively measure the textual coherence of subtitle segments. By combining these metrics, we can identify video segments that require visual information for better comprehension, thereby avoiding unnecessary visual analysis for most video segments.

B. Visual Content Analysis

VCA is designed to identify video segments that require visual information for comprehension. Initially, it leverages video encoding data to filter frames, minimizing local redundancy within the frame sequence. Next, an unsupervised learning-based filter is applied holistically across the entire video to retain only the most significant frames. Finally, MLLMs are used to analyze the key content of these frames, translating it into textual information for better understanding.

Frame Preprocessing based on Encoding Information. Considering the inherent dependencies and relative sizes among encoded frames in a video, we first eliminate frames to remove local redundancy using video encoding information. Specifically, we extract all I-frames from the original video $\mathcal{F}_o$, sort the P-frames associated with each I-frame by their sizes, and select the I-frame along with the top K most informative P-frames to form the candidate video frame set $\mathcal{F}_c$. This approach reduces the number of frames for subsequent processing and lowers computational costs. B-frames are excluded from $\mathcal{F}_c$ due to their reliance on bidirectional encoding between I-frames and P-frames.

Unsupervised Learning-based Frame Selector. Even after frame preprocessing, the subset $\mathcal{F}_c$ may still pose high computational costs for processing by MLLMs, especially for lengthy videos with substantial redundancy across long frame sequences. To address this, we designed an unsupervised learning-based frame selector to further reduce computational overhead and improve efficiency.

The unsupervised learning-based frame selector focuses on identifying frames essential for visual comprehension, transforming their key content into textual information for further analysis. Specifically, it employs a Residual Neural Network (**ResNet**) to extract deep features from frames and utilizes a self-attention mechanism to capture the temporal characteristics within the frame sequence. This process highlights and selects the most representative frames with significant dynamic variations. The loss function guiding the selection process is defined as follows:

$$L(\mathcal{F}_f) = R(\mathcal{F}_f) + D(\mathcal{F}_f), \tag{1}$$

where $L(\mathcal{F}_f)$ combines content diversity loss $R(\mathcal{F}_f)$ and temporal diversity loss $D(\mathcal{F}_f)$. The goal is to ensure that the selected frames effectively capture the video's overall content while preserving diversity within the set $\mathcal{F}_f$. The content diversity loss promotes a comprehensive representation of the video's key elements, while the temporal diversity loss encourages a well-dispersed temporal distribution of the selected frames across the video, enhancing their representativeness and diversity over time.

The content diversity loss, $R(\mathcal{F}_f)$, measures the similarity between the selected frame set $\mathcal{F}_f$ and the preprocessed frame set $\mathcal{F}_c$, as well as the diversity among frames within $\mathcal{F}_f$, as defined by the following equation:

$$\begin{aligned} R(\mathcal{F}_f) = &\frac{1}{|\mathcal{F}_f| \cdot |\mathcal{F}_c|} \sum_{i \in \mathcal{F}_f} \sum_{j \in \mathcal{F}_c} s_{ij} \\ &+ \frac{1}{|\mathcal{F}_f| \cdot (|\mathcal{F}_f| - 1)} \sum_{i \neq j} (1 - s_{ij}), \end{aligned} \tag{2}$$

where s_{ij} represents the cosine similarity between frame i and frame j. The first term evaluates the similarity between the selected frame set $\mathcal{F}_f$ and the preprocessed frame set $\mathcal{F}_c$, ensuring that the selected frames retain relevance to the preprocessed frames. The second term calculates the dissimilarity among frames within $\mathcal{F}_f$, encouraging internal diversity to enhance the representativeness of the selected set.

The temporal diversity loss, $D(\mathcal{F}_f)$, ensures that the selected frame set $\mathcal{F}_f$ is evenly distributed across the video timeline and distinguishable from the preprocessed set $\mathcal{F}_c$. By employing trigonometric positional encoding (denoted as $E(\cdot)$) to represent each frame's position within the video sequence, it quantifies the temporal differences between frames. This loss is designed to promote not only content diversity but also temporal dispersion, enabling the selected frames to effectively represent various segments of the video. The temporal diversity loss is defined as follows:

$$\begin{aligned} D(\mathcal{F}_f) = &\frac{1}{|\mathcal{F}_f| \cdot |\mathcal{F}_c|} \sum_{p_i \in \mathcal{F}_f, p_j \in \mathcal{F}_c} |E(p_i, k) - E(p_j, k)| \\ &- \frac{1}{|\mathcal{F}_f| \cdot (|\mathcal{F}_f| - 1)} \sum_{p_i, p_j \in \mathcal{F}_f, p_i \neq p_j} |E(p_i, k) - E(p_j, k)| \end{aligned} \tag{3}$$

The first term quantifies the temporal differences between the selected frames and the preprocessed frames. It ensures that $\mathcal{F}_f$ uniformly spans the entire video timeline, reflecting content from diverse time points. The second term encourages temporal dispersion among frames within $\mathcal{F}_f$, reducing clustering and enhancing the set's internal temporal diversity. This design ensures that the selected frame set comprehensively represents different parts of the video over time, effectively capturing its temporal dynamics.

By utilizing trigonometric positional encoding to represent frame positions, the model's ability to perceive temporal differences is enhanced, enabling it to better identify and

preserve critical moments while capturing long-term dependencies within the video.

Ultimately, this unsupervised learning-based video frame selector efficiently and accurately extracts key frames across diverse content categories, optimizing subsequent video analysis. Compared to uniform sampling of the original video, this approach reduces the number of tokens required for LLM analysis, significantly lowering computational costs. The detailed training methodology of the algorithm follows a similar approach to that outlined in [39].

C. Video Segment Summary Module

The VSSM processes subtitle information $\vec{t}$ from TCA and visual descriptions $\vec{d}$ from VCA to automatically segment the video and generate summaries for each segment, aiming to create a structured overview of the video content. To address the challenges of summarizing the entire video and determining segment timings, we draw inspiration from [40] and adopt the Chain of Thought (CoT) technique. This method organizes the inference process into three sequential steps: information integration, content segmentation, and summary output. By explicitly articulating the reasoning process, CoT enhances the quality of the generated output and provides a clear structure for the video content.

Information Integration. Given the subtitle information $\vec{t}$ and visual description $\vec{d}$, we first design a prompt (Appendix A) to extract detailed events from them. These extracted events form the unified narrative vector $\vec{r}$, which encapsulates the video's context and content.

Content Segmentation. Based on narrative vector $\vec{r}$, we divide the video into n segments using the prompt in Appendix B. Segmentation is guided by key transitions or new events within the narrative flow, providing a natural structure for generating detailed summaries of each segment.

Summary Output. Finally, a concise summary, title and stop time is generated for each video segment using the prompt in Appendix C.

D. Differentiated Video Chunk

Since each video segment generated from VSSM represents a distinct subtheme or narrative phase, the beginnings of these segments are regions where users are more likely to seek or jump to. Therefore, we designate the first K chunks of each key segments as *hp* chunks, ensuring they are prioritized for preloading or streaming in higher quality. A larger K value improves the content coverage but also increases prefetching overhead and limits bitrate flexibility. Experimentally, we have found that $K = 3$ provides a good balance between perceptual quality and system responsiveness in dynamic network environments. The remaining chunks are categorized as *lp* chunks. This differentiation helps minimize rebuffering time and prevents users from experiencing degraded quality when jumping to these segments.

VI. Adaptive Streaming Enhancement

The Adaptive Streaming Enhancement is performed using three key components designed to optimize video streaming performance and enhance the QoE: the Differentiated Encoding module, the Dynamic Buffer module, and the ABR Controller module. They are discussed next.

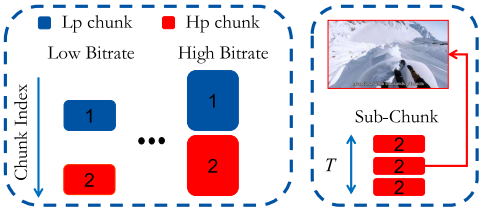


Fig. 7. Differentiated Encoding Module for DASH Video Streaming.

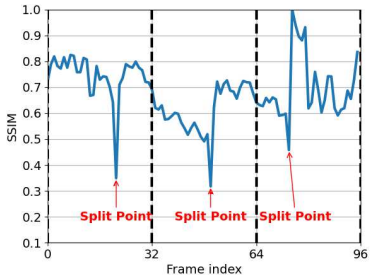


Fig. 8. Illustration of Split Points Selection.

A. Differentiated Encoding Module

The server-located Differentiated Encoding module encodes each video at multiple bitrates to support adaptive streaming. Initially, the video is encoded using the standard maximum video chunk length of $T = 4s$. However, high-bitrate *hp* chunks can take longer to download, and significant network fluctuations during this process may lead to rebuffering. To address this issue, we introduce the concept of a *sub-chunk*. Instead of encoding the video solely at the fixed 4-second chunk level, *hp* chunks are further divided and encoded into smaller units, referred to as sub-chunks. This finer-grained encoding approach is motivated by recent research showing that smaller video chunks enhance the system's ability to adapt to network fluctuations [41], [42], [43]. Each *hp* chunk is divided into M sub-chunks, c_i, enabling faster responses to changes in network throughput. The duration of the i-th sub-chunk is denoted as T_i^{sub}, with the total duration satisfying $T = \sum_{i=1}^{M} T_i^{sub}$. The resulting DASH video streaming workflow, incorporating the Differentiated Encoding module, is illustrated in Figure 7.

Selection of Split Points for Coding. The first thing is to determine how to segment *hp* chunks into sub-chunks. Assume the *hp* chunks are divided into equal-length sub-chunks, *i.e.,* $T_i^{sub} = \frac{T}{M}$. In this case, when the I-frame and its subsequent reference frame have low similarity, the total size of a video chunk might exceed that of encoding without sub-chunks, which is undesirable. To address this, we segment the *hp* chunks into sub-chunks based on their similarity.

As shown in Figure 8, the split points selection process consists of two steps. Assume a *hp* chunk contains N_f frames, *i.e.,* $\mathcal{F} = \{f_1, \ldots, f_{N_f}\}$. In *Step 1*, we divide $\mathcal{F}$ into $M-1$ segments at equal intervals, where the set of frames in the i-th

segment is denoted as $\mathcal{F}_i$. Then, f_1 and f_{N_f} are designated as the split points for the first and last segments, respectively. In *Step 2*, for each remaining segment $\mathcal{F}_i$, we select a frame f_{s_j} that is the least like the previous split point $f_{s_{j-1}}$ as the split point s_j. The frame similarity is measured using SSIM. The frames identified as split points are enforced as I-frames. Formally, f_{s_j} is defined as:

$$f_{s_j} = \begin{cases} f_1, & j = 1 \\ \min\limits_{f_k \in \mathcal{F}_j} \text{SSIM}(f_{s_{j-1}}, f_k), & j = 2, \ldots, M \\ f_{N_f}, & j = M+1 \end{cases} \tag{4}$$

Finally, the *hp* chunk is divided into M sub-chunks using the $M+1$ split points $\{s_1, \ldots, s_j, \ldots, s_{M+1}\}$ obtained.

Assigning Bitrate to Sub-Chunks. Next, the Differentiated Encoding Module assigns a bitrate to each sub-chunk. Assigning the same target bitrate R_{avg} directly to all sub-chunks would fail to maximize the average perceived quality of the *hp* chunks and would also result in significant quality variations across sub-chunks.

This issue arises because each sub-chunk contains only one I-frame, which consumes the majority of the allocated bitrate (while the remaining bitrate is distributed among P and B frames). Longer sub-chunks allocate more bitrate to P and B frames, leading to potentially higher quality compared to shorter sub-chunks. Consequently, it is necessary to assign different target bitrates to each sub-chunk.

To achieve this, the bitrate allocation is formulated as an optimization problem that maximizes the harmonic mean of the perceived quality of the sub-chunks while keeping the total size of all sub-chunks constant.

$$\max_{R_1,\ldots,R_i} \frac{M}{\sum_{i=1}^{M} \frac{1}{q(c_i)}} \tag{5}$$

$$s.t. \sum_{i=1}^{M} R_i T_i^{sub} = R_{avg} T \tag{6}$$

where $q(c_i)$ denotes the average VMAF of the sub-chunk c_i.

To solve this optimization problem, we employ the Simulated Annealing method [44], a probabilistic and greedy algorithm that approximates the global optimum. To enhance search efficiency, the minimum step size for adjusting the bitrate is set to $R_\Delta = 100$kbps, and the initial solution is defined as $S_{init} = \{R_i \mid R_i = R_{sub}, i = 1, \ldots, M\}$, indicating that the search begins with an equal bitrate allocation.

B. Dynamic Buffer Module

This module dynamically adjusts the ratio of buffer space reserved for prefetching future *hp* chunks.

Composition of the Dynamic Buffer. As illustrated in Figure 9, to ensure high quality of *hp* chunks without increasing the overall buffer size, we propose a dynamic buffer ratio strategy. The dynamic buffer adheres to the following two principles:

1) The buffer consists of a sequence buffer and a prefetch buffer. The total space allocated to these two buffers is fixed (*e.g.,* 60 seconds), but the ratio of space assigned to each can be adjusted dynamically.
2) The sequence buffer downloads video chunks in sequential order, while the prefetch buffer prefetches non-consecutive *hp* chunks (*i.e.,* jumps). When a video chunk in the prefetch buffer is about to play, it is automatically moved to the sequence buffer.

Updating the Dynamic Buffer Ratio. The dynamic buffer ratio plays a key role in performance. A too small sequence buffer may lead to rebuffering during sudden network jitter, while an excessively large sequence buffer can waste downloaded chunks when they are skipped and degrade the quality of *hp* chunks due to insufficient prefetch buffer space. By dynamically adjusting the length of sequence buffer l_s to accommodate network conditions and user viewing behavior, we can optimize resource allocation between the sequence buffer and the prefetch buffer.

Ideally, knowing precisely "when" and "where" a user will jump could prevent bandwidth waste. However, it is impossible to accurately predict if and when a user will jump to an *hp* chunk. Thus, our approach does not rely on predicting user jumps. Instead, we control the length of the sequence buffer to ensure video quality does not significantly degrade during prefetching. Specifically, the length of the sequence buffer l_s is calculated as:

$$l_s = \lceil (s_{hp} + R_{last} \times T) / \hat{C}_{avg} \rceil, \tag{7}$$

where s_{hp} denotes the size of the next *hp* chunk, R_{last} is the bitrate of the previous chunk, and $\hat{C}_{avg}$ is the estimated average throughput. To prevent the sequence buffer from being too short to sustain playback, l_s is always set to be greater than half of the total buffer size. Additionally, when the prefetch buffer is sufficient to support streaming the *hp* chunk at the highest bitrate, l_s is set to the remaining portion of the total buffer, *i.e.,* prefetching is paused. This strategy enables JumpDASH to maintain strong performance during long-term network fluctuations, even when users do not jump.

C. ABR Controller Module

This module determines whether the client should sequentially download the next video chunk or "jump" to prefetch a later (non-sequential) *hp* chunk. The decision is based on the sequential buffer length l_s and the estimated throughput.

Actions. Traditional ABR algorithms do not account for users' nonlinear viewing patterns, and their action spaces lack support for jump prefetching. To address this limitation, we expand the ABR action space. We define the i-th step decision of ABR as o_i, where $o_i \in \mathcal{R} \cup \{P\}$. Here, P denotes prefetching the first sub-chunk into the prefetch buffer, while $\mathcal{R}$ represents the bitrate set of the chunks. When the buffer size $B_i \geq l_s$, we set $o_i = P$, indicating that the ABR controller decides to prefetch the highest bitrate version of the sub-chunks. Conversely, when $B_i < l_s$, the controller makes a sequential bitrate decision, with $o_i \in \mathcal{R}$.

Inputs. After completing the download of each chunk i, the system provides state inputs $S_i = (\boldsymbol{C_i}, \boldsymbol{s_i}, \boldsymbol{b_i}, \boldsymbol{d_i}, v_i, pb_i, ps_i, r_i)$ to the model. Here, $\boldsymbol{C_i}$ represents the throughput for the past L chunks, $\boldsymbol{s_i}$ indicates the sizes of the next sequential chunk, $\boldsymbol{b_i}$ and

$\boldsymbol{d_i}$ denote the past buffer occupancy and download times, respectively, v_i is the average bitrate of the previously downloaded chunk, pb_i represents the prefetch buffer size, ps_i is the size of the next prefetched sub-chunk, and r_i signifies the normalized count of remaining video chunks.

Reward Function. The reward function is defined to incorporate both sequential download and prefetching decisions:

$$R_i = \begin{cases} q(R_i) - \alpha_1 T_i - |q(R_i) - q(R_{i-1})|, o_i \in \mathcal{R} \\ \alpha_2 q(R_{max}) - \alpha_3 \max(0, B_{\min} - B_{cur}), o_i = P \end{cases} \tag{8}$$

In the sequential download decision, represented by $o_i \in \mathcal{R}$, the reward R_i consists of the following components:

- $q(R_i)$: The quality of the current bitrate selection R_i, reflecting the quality of the video chunk.
- T_i: The rebuffering time for the current chunk, which negatively impacts the QoE.
- $|q(R_i) - q(R_{i-1})|$: The absolute difference in quality between consecutive chunks, with a weighting factor α_1 that balances the importance of video quality, the impact of rebuffering, and playback smoothness.

For the prefetching decision, represented by $o_i = P$, the reward function prioritizes the following:

- $q(R_{\max})$: The quality associated with prefetching at the highest bitrate, $R_{\max}$, reflecting the benefit of obtaining high-quality video content in advance.
- A penalty mechanism, $\max(0, B_{\min} - B_{\text{cur}})$, which is activated if the current buffer size B_{cur} drops below the minimum required buffer size $B_{\min}$. This penalizes situations where prefetching risks depleting the buffer and causing playback interruptions.

The weights α_2 and α_3 adjust the relative importance of achieving high-quality prefetching and mitigating the impact of buffer underflow, respectively. This comprehensive approach to defining R_i dynamically adapts the streaming strategy, prioritizing optimal video quality while ensuring smooth and uninterrupted playback. By balancing high-quality prefetching with maintaining sufficient buffer levels to avoid rebuffering events, it effectively maximizes the overall QoE.

Training Method. The network architecture consists of an Actor and a Critic, each processing input state data through separate pathways. Sequence data is handled by linear layers that simulate convolutions, while scalar data is processed using fully connected (**FC**) layers. The outputs from these pathways are merged and further processed through additional FC layers.

In the Actor module, the integrated output is passed through an FC layer that applies a softmax function to produce action probabilities. Meanwhile, the Critic module outputs a single value estimate through an FC layer, representing the expected return of the current state. During training with PPO, distinct computational and optimization strategies are applied to the Critic and Actor networks, ensuring effective learning of the policy and value functions within the environment.

Synchronous Parallel Training. Similar to Pensieve, we employ a parallel training methodology to accelerate the training process by using multiple agents. However, unlike Pensieve [4], which uses asynchronous updates in its A3C algorithm, we opt for synchronous updates. This shift to synchronous updating aims to harmonize the training process by ensuring that updates across all parallel environments are coordinated and consistent, thereby accelerating convergence. By adopting synchronous updates, the algorithm's ability to explore a broader spectrum of states is enhanced, and the training timeline is also shortened.

VII. Experimental Assessment

Our experiments are divided into five parts with following objectives: (*i*) Can JumpDASH comprehend the video effectively and generate accurate text summaries with fewer tokens (Section § VII-A)? (*ii*) How is the performance of JumpDASH compared to other ABR schemes under various simulated network conditions (Section § VII-B)? (*iii*) How is the performance of JumpDASH compared to other ABR algorithms in real-world network conditions (Section § VII-C)? (*iv*) How is the impact of each component on overall performance (Section § VII-D)? and (*v*)How is the performance of JumpDASH with different user viewing behavior and maximum buffer lengths (Section § VII-E and § VII-F)? (*vi*) How is the system overhead of JumpDASH (Section § VII-G)?

A. Performance of Low-Cost Video Understanding

Experiment Details. Experiment A is conducted on an Ubuntu 20.04 LTS system equipped with eight NVIDIA GeForce V100 GPUs. We utilized the LLaMA-VID [34] model, which has a parameter size of 7B. This open-source visual MLLM has demonstrated remarkable performance across multiple video datasets and is capable of understanding long video content, making it the visual content understanding module for this study. In the VSSM module, due to the often lengthy nature of video subtitles, we employed an open-source large language model, ChatGLM3 [45], with a 6B parameter size.

Design of Experiment A.1. We selected the VideoInstruct100K video dataset from Video-ChatGPT [46], which contains 500 videos without subtitles and 2000 related video questions. This dataset is used to evaluate the MLLM's capabilities in five aspects: accuracy of information, understanding of details, context comprehension, temporal understanding, and consistency. It is employed to systematically assess the impact of the frame selection ratio P on the MLLM's ability to understand the entire video, as well as to determine the optimal value for P.

The evaluation process follows the approach outlined in Video-ChatGPT [46]. Specifically, first we generated answers to the corresponding questions in the video dataset using MLLM when the video is sampled every 15 frames (*i.e.,* uniform sampling) as a baseline. Next, we sampled video frames with different frame selection ratios P using the unsupervised learning-based frame selector described in Section § V-B to generate answers. Subsequently, ChatGPT-3.5 was employed to assess how well each aspect (*e.g.,* information accuracy) sits between the ground truth answers from the dataset and the answers generated from the sampled video frames (including both uniform sampling and our frame selector) on a scale from 0 (lowest match) to 5 (highest match). Finally, we compute

the relative score change by subtracting the score of uniform sampling from the score of our frame selector.

Design of Experiment A.2. We randomly select 20 videos with subtitle information from YouTube to assess the overall effectiveness of the low-cost video understanding strategy in JumpDASH. Three methods were set up for comparison to validate the effectiveness of the frame selection-based low-cost video text summary module in comprehending the video content with reduced computational cost:

- Method 1: Analysis of the subtitle file only using the VSSM module;
- Method 2: Uniform sampling of the entire video to obtain a complete video description using LLaMA-VID, followed by the analysis of both the subtitle file and video description using the VSSM module;
- Method 3: Analysis of the entire video using the frame selection-based low-cost video understanding strategy (including TCA, VCA and VSSM).

We first manually segmented the 20 videos and wrote corresponding summaries and titles to serve as ground truth for evaluation. Then, we applied the three video understanding methods to generate summaries and titles for the segments. Finally, OpenAI ChatGPT-3.5 and ChatGPT-4.1 were used to score the degree of match between the ground truth and the outputs of each method on a scale from 0 (lowest match) to 5 (highest match), demonstrating the effectiveness of each video understanding approach. We did not report the segment accuracy separately as the segments are primarily used to help users navigate the video. Therefore, precise segment boundaries are not important provided that the content is meaningfully summarized and understood. Additionally, we recorded the total token counts of subtitle text and video frame inputs to assess the cost-effectiveness of the different methods.

Design of Experiment A.3. Experiment A.3 follows the same setup as Experiment A.2, except that we randomly select 10 videos from each of the four video categories (*i.e., film*, *documentary*, *sports*, and *news*) and evaluate them using ChatGPT-3.5. This experiment is designed to assess the video understanding effectiveness of the low-cost video understanding strategy in JumpDASH across different categories.

Results of Experiment A.1. From the analysis of the results of the video frame filter experiment, as shown in Table I, the following conclusions can be drawn:

- Adjusting the video frame selection ratio P has an impact on various evaluation metrics of the model, but importantly the magnitude of this impact is limited, as the relative changes fall within the $[-0.1, 0.1]$ range, while the relative scores range from $[-5, 5]$. This suggests that our proposed frame selection strategy can enhance computational efficiency without significantly compromising the quality of understanding.
- When the P value is moderately increased (such as to 0.05 and 0.10), the temporal understanding ability is enhanced. This is due to the video frame filter effectively removing redundancy in the video frames It allows the model to focus on information-rich frames, thereby optimizing the understanding quality.

TABLE I

EVALUATION RESULTS FOR THE EFFECT OF DIFFERENT P SETTINGS ON THE MLLM (RELATIVE CHANGE)

	Parameter Setting		
Metric	P=0.01	P=0.05	P=0.10
Information Accuracy	-0.09	+0.01	+0.02
Detail Understanding	-0.02	+0.02	+0.02
Context Understanding	-0.05	-0.01	-0.01
Temporal Understanding	+0.05	+0.07	+0.08
Consistency	-0.01	-0.00	+0.01

TABLE II

COST AND VIDEO UNDERSTANDING EFFECTIVENESS FOR NARRATED VIDEOS

Method	Token Number	Score (GPT-3.5)	Score (GPT-4.1)
Method 1	6,709	3.81	2.21
Method 2	5,065,534	3.78	2.16
Method 3	10,062	4.02	2.72

- At lower P values (such as 0.01), information accuracy decreases significantly, reflecting that extensive filtering weakens the model's ability to capture key information.
- Regarding consistency, changes under various parameter settings are not significant, indicating that video frame selection has little impact on the model's consistency.

These results indicate that with the video frame filter proposed, we can effectively reduce the tokens used by the current MLLMs while still performing well in visual understanding tasks. Considering the performance difference between $P = 0.05$ and $P = 0.10$ is negligible, we choose to use $P = 0.05$ due to its lower computational cost.

Results of Experiment A.2. Table II presents the average input token number and video understanding scores (evaluated by both ChatGPT-3.5 and ChatGPT-4.1) for the three methods on a custom YouTube video dataset. The token number for Method 1 serves as a baseline, whereas the token number for Method 3 is only slightly higher than Method 1, *i.e.,* only 50% higher than that of Method 1. Despite this, Method 3 achieved a video understanding score of 4.02, surpassing Method 1's score of 3.81, clearly demonstrating that integrating visual information enhances understanding. Compared to Method 2, the token number of Method 3 is much smaller, indicating that the proposed TCA and VCA modules significantly reduce resource consumption while providing a higher video understanding score (4.02 compared to 3.78). We also evaluate the video understanding score using the newer ChatGPT-4.1 model. The results show that while GPT-4.1 assigns lower scores than GPT-3.5, the overall trends in video understanding performance remain consistent across both models. Specifically, Method 2 achieves scores comparable to Method 1, and Method 3 attains the highest score.

Results of Experiment A.3. Table III presents the average input token count and video understanding scores for the three methods across four video categories. Compared to Method 1, Method 3 achieves comparable or higher video understanding scores with only a moderate increase in cost. For instance, in the *news* category, Method 3 outperforms Method 1 by

TABLE III
COST AND VIDEO UNDERSTANDING EFFECTIVENESS FOR DIFFERENT CATEGORIES

Category	Method	Token Number	Score
Film	Method 1	1.42×10^4	3.82
	Method 2	8.20×10^6	4.28
	Method 3	1.55×10^4	3.59
Documentary	Method 1	9.88×10^3	3.86
	Method 2	6.24×10^6	4.50
	Method 3	1.16×10^4	4.33
Sports	Method 1	1.02×10^4	2.64
	Method 2	3.93×10^6	3.15
	Method 3	9.95×10^3	3.10
News	Method 1	9.73×10^3	3.55
	Method 2	5.22×10^6	4.12
	Method 3	1.10×10^4	3.96

0.41 in score, while incurring only a 13% higher cost. In comparison to Method 2, Method 3 significantly reduces cost while maintaining similar performance. For example, in the *sports* category, Method 3's score is just 0.05 lower than that of Method 2, but the cost is reduced by 99.7%. These findings are consistent with the results shown in Table II.

B. Performance Under Simulated Network Conditions

Test setup. To evaluate JumpDASH, the video server is implemented using *Nginx*, and the client player is based on *Python*. We configure the round-trip time to 80 ms and use Mahimahi [47] to simulate the bandwidth between the video server and the client by replaying network traces. Meanwhile, the Pensieve simulator [4] is modified for training learning models and parameter validation.

Network trace. We use 3G [48], Oboe [49], and 4G [50] traces to test the performance of JumpDASH under various network conditions. These traces are collected in different scenarios and contain throughput information recorded per second for different durations. The mean and standard deviation (represented as $\mu \pm \sigma$) of the throughput values in 3G, Oboe, and 4G are 1.67 ± 0.99 Mbps, 2.84 ± 1.60 Mbps, and 5.07 ± 4.08 Mbps, respectively. We randomly select 70% of the traces for training and 30% for testing.

Video dataset. To test the performance of JumpDASH, we utilized the video dataset mentioned earlier. Within this dataset, we set the first $K = 3$ chunks of each segment as *hp* chunks and the number of sub-chunks $M = 4$. For fairness, we used the same settings as [4], [16], and [18], which are encoded as [300, 750, 1200, 1850, 2850, 4300] Kbps. Each video duration is limited to 360 seconds. We utilized FFmpeg to encode all videos using x264 and convert them to DASH format.

User setting. Each individual user may choose different *hp* chunks. Each user selects a different time node to play the *hp* chunk. Therefore, we randomly generate 5 time points between the first video chunk and the first *hp* chunk, representing when 5 different users jump to *hp* chunks.

ABR scheme baselines. In the experiments, we compare our scheme with the following existing ABRs: 1) *BOLA* [5]: A near-optimal buffer-based algorithm that determines the bitrate based only on the state of the buffer, used in Dash.js; 2) *RobustMPC* [6]: The video chunk bitrate is selected by predicting the throughput and using control theory; 3) *Pensieve* [4]: The A3C reinforcement learning method is used to select the video chunk bitrate; 4) *Comyco* [18]: A content-aware DRL algorithm with expert policies; and 5) *Merina* [16]: A meta-reinforcement learning-based neural adaptive bitrate streaming algorithm.

TABLE IV
ABR ALGORITHM COMPARISON ON THE 3G DATASET

ABR Scheme	QoE	*hp* Quality	*hp* Rbf. Time	All Rbf. Time
RobustMPC	0.39	0.65	0.50	0.23
BOLA	0.23	0.43	0.51	0.19
Pensieve	0.22	0.71	0.73	0.22
Comyco	0.14	1.36	1.54	0.28
Merina	0.27	0.72	0.70	0.22
JumpDASH	0.51	2.16	0.48	0.11

hp Quality: *hp* Chunk Quality; Rbf.: Rebuffering

TABLE V
ABR ALGORITHM COMPARISON ON THE OBOE DATASET

ABR Scheme	QoE	*hp* Quality	*hp* Rbf. Time	All Rbf. Time
RobustMPC	1.55	1.17	0.37	0.17
BOLA	1.02	0.74	0.39	0.14
Pensieve	1.12	1.44	0.76	0.19
Comyco	1.27	2.13	1.25	0.27
Merina	1.27	1.27	0.63	0.18
JumpDASH	1.88	3.05	0.18	0.08

TABLE VI
ABR ALGORITHM COMPARISON ON THE 4G DATASET

ABR Scheme	QoE	*hp* Quality	*hp* Rbf. Time	All Rbf. Time
RobustMPC	2.46	1.99	0.36	0.13
BOLA	1.90	1.47	0.37	0.10
Pensieve	2.17	2.32	0.81	0.15
Comyco	2.20	3.37	1.73	0.25
Merina	2.36	2.17	0.62	0.14
JumpDASH	2.80	3.97	0.34	0.08

QoE metric. To ensure fairness, we adopt a widely used QoE setting [4], [6], [16], where $q(R_i) = R$ represents the video quality for an average L-second length video chunk, with $\beta = 4.3$, as described by the following equation:

$$QoE = \sum_{i \in \mathcal{W}} q\left(R_i\right) - \beta \sum_{i=1}^{N} T_i - \sum_{i \in \mathcal{W}} \left| q\left(R_{i+1}\right) - q\left(R_i\right) \right| \quad (9)$$

$\mathcal{W}$ represents the set of video chunks that have been watched. Differently, we only calculate the quality gain and smoothness penalty for the video chunks that users have watched, instead of all downloaded video chunks.

Results of Experiment B. Tables IV, V, and VI present performance of several ABR algorithms across three datasets: 3G, Oboe, and 4G. *hp* rebuffering time refers to the rebuffering time encountered by users while watching *hp* chunks.

From the results, we observe that JumpDASH consistently outperforms other algorithms across all key performance indicators on the 3G, Oboe, and 4G datasets. Notably, in terms of

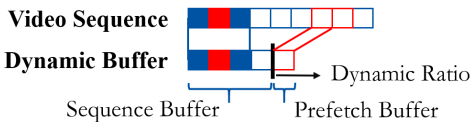


Fig. 9. The composition of dynamic buffer.

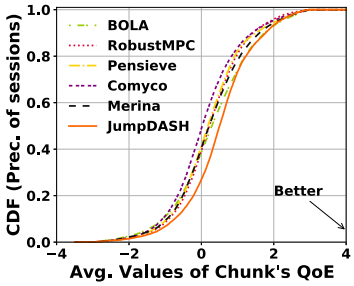


Fig. 10. CDF of QoE in a 3G network.

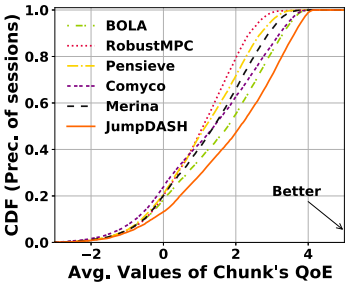


Fig. 11. CDF of QoE in an Oboe network.

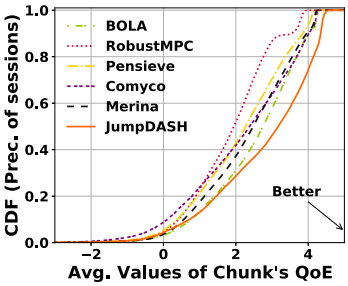


Fig. 12. CDF of QoE in a 4G network.

average QoE, JumpDASH shows an improvement of approximately 30.77%-264.29% over the other ABR algorithms in the 3G dataset, 21.29%-84.31% in the Oboe dataset, and 13.82%-47.37% in the 4G dataset. These significant improvements highlight JumpDASH's superior ability to optimize user QoE under diverse network conditions.

Figures 10, 11, and 12 present the CDFs of QoE, illustrating the distribution of quality experiences across different network environments. Specifically, we simulate the streaming of the same video on each testing trace, referred to as a session. For each session, we calculate the average QoE value of the video chunks and depict the cumulative distribution of all sessions. Notably, in over 90% of the sessions, JumpDASH consistently exhibits the highest QoE.

Furthermore, across all three datasets, JumpDASH effectively enhances the *hp* chunk quality and reduces *hp* rebuffering time, while maintaining a relatively low total rebuffering time. This is particularly evident in the Oboe dataset, where JumpDASH's *hp* chunk quality is significantly higher than that of its closest competitor, while still achieving the lowest *hp* rebuffering time. These results highlight JumpDASH's strengths in scenarios involving nonlinear viewing patterns, ensuring high *hp* chunk quality without frequent rebuffering.

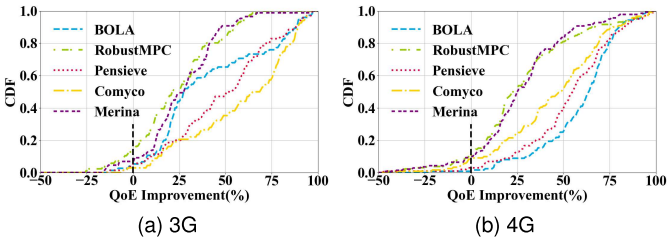


Fig. 13. QoE improvement of JumpDASH over other methods.

Moreover, we observe that reinforcement learning-based ABR algorithms generally underperform in nonlinear viewing scenarios. Notably, the Comyco algorithm, which utilizes inverse reinforcement learning, demonstrates markedly poor performance. This suggests that without specific modifications and adaptations, such algorithms may lead to suboptimal user experiences when viewers manipulate the playback timeline, as commonly occurs in real-world usage.

To further demonstrate the improvements brought by JumpDASH, we conducted additional experiments where baseline ABR methods were augmented with JumpDASH's sub-chunk encoding and jump prefetching mechanisms. The results, shown in Figure 13, indicate that JumpDASH outperforms the best-performing baselines in 93.10% of 3G and 90.01% of 4G network traces. These results highlight the effectiveness of JumpDASH even when the baselines are equipped with the same fine-grained bitrate control capabilities.

C. Performance Under Real-World Network Conditions

Test setup. In the real-world experiment, the setup is similar to Experiment B, with the following differences:

- On the client side, we modified the open-source video player ffplay to support jump prefetching for downloading video chunks and a dynamic buffer module. Additionally, an interface program was implemented to receive ABR commands in a Python environment.
- We conducted tests using a real 4G cellular network and a public Wi-Fi network on campus, with a mean and standard deviation of recorded throughput values of 5.01 ± 0.31 Mbps and 3.53 ± 0.54 Mbps, respectively.
- Due to the limitations of real-network testing, where it is not possible to fast-forward network speed during playback acceleration, we selected the heuristic-based RobustMPC, the reinforcement learning-based methods Merina, and our JumpDASH for comparison.

Results of Experiment C. As shown in Tables VII and VIII, JumpDASH consistently outperforms other ABR schemes in both real-world 4G and WiFi environments. This

TABLE VII
COMPARISON OF ABR ALG. IN A REAL 4G NETWORK

ABR Scheme	QoE	*hp* Quality	*hp* Rbf. Time	All Rbf. Time
RobustMPC	2.45	3.21	0.15	0.11
Merina	2.23	3.43	0.18	0.10
JumpDASH	2.91	4.01	0.04	0.03

TABLE VIII
ABR ALGORITHM COMPARISON IN REAL WIFI NETWORK

ABR Scheme	QoE	*hp* Quality	*hp* Rbf. Time	All Rbf. Time
RobustMPC	2.25	3.16	0.26	0.19
Merina	2.13	3.34	0.27	0.18
JumpDASH	2.59	3.72	0.08	0.07

TABLE IX
COMPARISON OF ABLATION STUDY RESULTS FOR JUMPDASH VERSIONS ON ABR PERFORMANCE

ABR Scheme	QoE	*hp* Quality	*hp* Rbf. Time	All Rbf. Time
JumpDASH-v0	2.13	2.08	0.60	0.12
JumpDASH-v1	2.47	3.20	0.59	0.12
JumpDASH-v2	2.80	3.97	0.34	0.08

highlights JumpDASH's ability to optimize streaming experiences across diverse network conditions. Specifically, in the high-throughput and stable real 4G conditions, JumpDASH significantly improves user QoE by reducing *hp* rebuff times and increasing *hp* chunk quality through its preemptive prefetching strategies. This also suggests that as network technologies advance, JumpDASH is likely to demonstrate greater superiority in scenarios where network throughput exceeds video bitrate requirements, compared to other ABR algorithms.

D. Ablation Study

Test setup. This experiment is configured similarly to Experiment B, with the main difference being the comparison of the performance of three algorithms on the 4G dataset.

- JumpDASH-v0, which does not perform *hp* chunk prefetching;
- JumpDASH-v1, which performs prefetching of *hp* chunks but does not implement a differentiated encoding strategy;
- JumpDASH-v2, which implements the complete scheme.

Results of Experiment D. The data from Table IX shows that JumpDASH-v2 significantly improves the quality of *hp* chunks compared to JumpDASH-v0. Specifically, the *hp* chunk quality in JumpDASH-v2 is 3.97, a notable improvement from 2.08 in JumpDASH-v0, representing an approximate increase of 90.86%. Additionally, compared to JumpDASH-v1, JumpDASH-v2 effectively reduces *hp* rebuffering times from 0.59 to 0.34. This reduction, approximately 42.37%, suggests that the strategy of varying the length of encoded *hp* chunks in JumpDASH-v2 plays a significant role in minimizing rebuffering times caused by prefetching operations. The results also show that JumpDASH-v2 achieves the highest QoE across all datasets, with a value of 2.80, compared to 2.13 for JumpDASH-v0 and 2.47 for JumpDASH-v1.

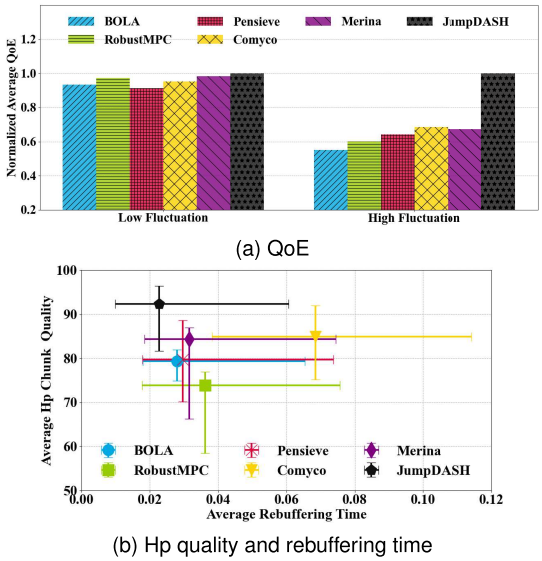


Fig. 14. Performance comparison when the user views videos sequentially.

These results confirm the effectiveness of the *hp* chunk prefetching and differentiated encoding strategy in JumpDASH-v2, which improves *hp* chunk quality, reduces rebuffering time, and enhances overall QoE. This makes JumpDASH-v2 an optimal solution for streaming under diverse network conditions.

E. Impact of User Viewing Behaviors

Test Setup. In this section, we evaluate the performance of JumpDASH when users watch videos sequentially, *i.e.,* without jumping. Specifically, we divide the 4G traces into low fluctuation and high fluctuation based on the coefficient of variation of bandwidth (c_v). c_v is defined as $c_v = \frac{\delta}{\mu}$, where δ and μ are the standard deviation and mean of the trace throughput, respectively. Low fluctuation ($c_v \in [0, 50\%]$) indicates short-term bandwidth variation, while high fluctuation ($c_v \in [100\%, 300\%]$) reflects long-term bandwidth variation. Similar to Experiment B, we compare JumpDASH against five existing ABR algorithms: *BOLA*, *RobustMPC*, *Pensieve*, *Comyco*, and *Merina*.

Results of Experiment E. Figure 14a shows the QoE of the six approaches under different bandwidth fluctuation conditions when users view videos sequentially. The results indicate that JumpDASH outperforms other ABRs in both settings. Under low bandwidth fluctuation, the QoE differences between ABRs are relatively small (<8%), whereas under high bandwidth fluctuation, the QoE advantage of JumpDASH becomes more pronounced. Specifically, Figure 14b presents the *hp* chunk quality and rebuffering time—two key factors influencing QoE—across the compared approaches. JumpDASH consistently achieves the highest *hp* chunk quality and the lowest rebuffering time. Compared to the best-performing baseline (BOLA), JumpDASH reduces rebuffering time by 18.97%. These results demonstrate that JumpDASH delivers superior performance even when users watch videos without jumping.

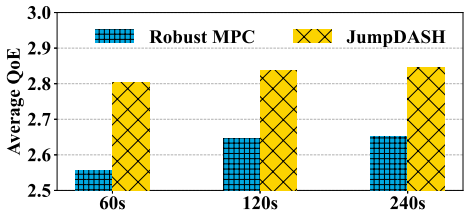


Fig. 15. Performance comparison of JumpDASH and Robust MPC under different buffer lengths.

F. Impact of Buffer Length

Test setup. This experiment is set up similarly to Experiment B. The main difference in Experiment F is that it primarily tests the QoE comparison between Robust MPC and JumpDASH at 60s, 120s, and 240s to explore JumpDASH's performance under different maximum buffer lengths. In this experiment, only the 4G dataset is used.

Results of Experiment F. Figure 15 compares QoE between Robust MPC and JumpDASH across different maximum buffer lengths. It is clear that JumpDASH consistently maintains a significantly higher QoE across all three buffer sizes. This highlights JumpDASH's strong adaptability to buffer size, effectively ensuring an optimal viewing experience for users, even with variations in buffer lengths.

G. System Overhead

Training Cost. JumpDASH involves training two modules: (*i*) unsupervised learning-based frame selector (Section § V-B), and (*ii*) ABR controller module (Section § VI-C). On our server equipped with eight NVIDIA GeForce V100 GPUs, training the frame selector on the VideoInstruct100K dataset takes approximately 2 hours. The ABR controller module, trained with 16 parallel agents, requires about 1 hour. Since both modules are trained only once, the overall training cost is low.

Processing Delay. In JumpDASH, only the ABR controller module operates online during each viewing session, while all other modules are processed offline with a one-time cost. The ABR controller module employs a lightweight DRL model for decision-making, with an inference latency of 3–10 ms on standard client devices, ensuring no playback delay. For a 360-second video, the processing latency of the offline modules is as follows: (*i*) TCA: 0.5s, (*ii*) VCA: 9s, (*iii*) VSSM: 30s, and (*iv*) Differentiated Encoding Module: 26s. In total, the offline processing latency amounts to 66 seconds. Since these modules are executed only once per video on the server side prior to playback, the overhead is acceptable.

Overhead of Sub-Chunk Encoding. While sub-chunk encoding improves bitrate adaptability and reduces quality fluctuations, it introduces trade-offs in processing and storage. (*i*) *Processing Overhead:* Each *hp* chunk is divided into 4 sub-chunks, with each sub-chunk encoded into 6 bitrate versions. This increases the total number of encoding tasks. For example, in a 360-second video with 90 chunks, the number of encoding jobs rises from 540 (90 chunks $\times$6 bitrates) to 864 (18 *hp* chunks $\times$4 sub-chunks $\times$6 bitrates + 72 standard chunks $\times$6 bitrates). (*ii*) *Storage Cost:* Traditional DASH stores 6 versions per chunk, whereas our approach stores 6 versions per sub-chunk for *hp* chunks. For each *hp* chunk split into 4 sub-chunks, this results in a 2–2.5$\times$ increase in storage for those chunks. Despite this, the process is entirely offline and parallelizable. With GPU acceleration, the end-to-end processing time remains manageable. Moreover, this overhead is limited to *hp* chunks only, which comprise just 30–40% of the total video content. For the remaining chunks, standard encoding is retained.

VIII. Conclusion

This paper introduces JumpDASH, a significant advancement over traditional ABR algorithms that improves user video streaming experience. Unlike existing ABR systems, which assume sequential video playback and overlook user-driven navigation through video content, JumpDASH supports the highly dynamic viewing habits of modern users. By incorporating a low-cost video text summary module, JumpDASH enables users to quickly understand and locate segments of interest within videos, enhancing user engagement and satisfaction. Furthermore, an innovative jump prefetching technique is introduced to optimize network resource allocation by preloading the video chunks of high user interest. Experimental results demonstrate that JumpDASH significantly improves viewer QoE by 13.82% to 264.29% compared to existing ABR strategies. These show that JumpDASH offers a highly valuable user-centric solution that aligns with the nonlinear viewing preferences of today's digital audience.

Appendix
Prompts for the VSSM Module

A. Prompt for Information Integration

Given a complete set of video subtitles and description, your task is to analyze the text and identify detailed, sequential events throughout the video. Please ensure that these events are in chronological order, reflecting the progression of the video's narrative. Your output should provide a granular breakdown of the content, focusing on key developments, character actions, and significant dialogue or scenes. This detailed event list will serve as a foundational analysis for further video segmentation. The subtitle text to be analyzed is provided below: {subtitle text}. The video description to be analyzed is provided below: {video description}.

The {subtitle text} and {video description} are replaced with the subtitle information $\vec{t}$ and visual descriptions $\vec{d}$.

B. Prompt for Content Segmentation

Analyze the provided detailed event to identify distinct changes in the theme or narrative within the video. Your objective is to segment the video into several distinct segments. Focus on the video in its entirety, ensuring that no segment is overlooked. Each segment should encapsulate a unique subtheme or a specific phase in the narrative's evolution. This segmentation task is crucial for breaking down the video's complex content into digestible and coherent segments, facilitating an enhanced viewing experience. Ensure that each

segment you identify represents a logical transition, capturing the essence of the video's evolving storyline or thematic progression. Aim for clarity and precision in demarcating the boundaries of each segment to aid viewers in grasping the nuanced content without being overwhelmed. The video description to be analyzed is provided below: {narrative vector}.

The above prompt is used to segment the video, then the following prompt is used to obtain the number of video segments for later processing.

Analyze the segmented video content you provided earlier and indicate the total number of segments identified. Format your response as a solitary number without additional text or context. For example, simply respond with "4" if you identified 4 segments.

C. Prompt for Summary Output

Provide a concise summary for segment {n}, capturing the essence and distinguishing features of this segment in relation to the overall video narrative. Ensure the summary is compact and focused, offering valuable insight into the segment's content and purpose. Respond only with the summary, without additional explanations or text.

Create a concise and descriptive title for segment {n}, encapsulating its main theme or content succinctly. Your response should only include the title, free from any supplementary text or context.

Identify the stop time for segment {n} and provide it in a clear, standardized format such as "HH:MM:SS". Ensure that your response contains only the time information, formatted correctly, without additional commentary or details.

This prompt is executed n times to generate the results for all segments.

REFERENCES

[1] DemandSage.*Video Marketing Statistics 2025*. Accessed: Jul. 8, 2025. [Online]. Available: https://www.demandsage.com/video-marketing-statistics/

[2] Insivia.*Video Marketing Statistics You Must Know in 2025*. Accessed: Jul. 8, 2025. [Online]. Available: https://www.insivia.com/video-marketing-statistics-you-must-know-in-2025

[3] *Dash Industry Forum*. Accessed: Aug. 25, 2024. [Online]. Available: https://dashif.org/

[4] H. Mao, R. Netravali, and M. Alizadeh, "Neural adaptive video streaming with pensieve," in *Proc. Conf. ACM Special Interest Group Data Commun.*, Aug. 2017, pp. 197–210.

[5] K. Spiteri, R. Urgaonkar, and R. K. Sitaraman, "BOLA: Near-optimal bitrate adaptation for online videos," *IEEE/ACM Trans. Netw.*, vol. 28, no. 4, pp. 1698–1711, Aug. 2020.

[6] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli, "A control-theoretic approach for dynamic adaptive video streaming over HTTP," in *Proc. ACM Conf. Special Interest Group Data Commun.*, Aug. 2015, pp. 325–338.

[7] F. Y. Yan et al., "Learning in situ: A randomized experiment in video streaming," in *Proc. 17th USENIX Symp. Networked Syst. Design Implement. (NSDI)*, 2019, pp. 1–12.

[8] Z. Wang et al., "MultiLive: Adaptive bitrate control for low-delay multi-party interactive live streaming," *IEEE/ACM Trans. Netw.*, vol. 30, no. 2, pp. 923–938, Apr. 2022.

[9] J. Jiang, V. Sekar, and H. Zhang, "Improving fairness, efficiency, and stability in HTTP-based adaptive video streaming with festive," *IEEE/ACM Trans. Netw.*, vol. 22, no. 1, pp. 326–340, Feb. 2014.

[10] Z. Meng et al., "Practically deploying heavyweight adaptive bitrate algorithms with teacher–student learning," *IEEE/ACM Trans. Netw.*, vol. 29, no. 2, pp. 723–736, Apr. 2021.

[11] R. Rassool, "VMAF reproducibility: Validating a perceptual practical video quality metric," in *Proc. IEEE Int. Symp. Broadband Multimedia Syst. Broadcast. (BMSB)*, Jun. 2017, pp. 1–2.

[12] M. Sun, A. Farhadi, and S. Seitz, "Ranking domain-specific highlights by analyzing edited videos," in *Proc. 13th Eur. Conf. Comput. Vis.-ECCV*, Sep. 2014, pp. 787–802.

[13] W. Li, J. Huang, S. Wang, S. Liu, and J. Wang, "DAVS: Dynamic-chunk quality aware adaptive video streaming using apprenticeship learning," in *Proc. IEEE Global Commun. Conf.*, Taipei, Taiwan, Dec. 2020, pp. 1–6.

[14] A. Garcia del Molino and M. Gygli, "PHD-GIFs: Personalized highlight detection for automatic GIF creation," in *Proc. 26th ACM Int. Conf. Multimedia*, Oct. 2018, pp. 600–608.

[15] S. Kim, H. Oh, and C. Kim, "Eff-HAS: Achieve higher efficiency in data and energy usage on dynamic adaptive streaming," *J. Commun. Netw.*, vol. 20, no. 3, pp. 325–342, Jun. 2018.

[16] N. Kan, Y. Jiang, C. Li, W. Dai, J. Zou, and H. Xiong, "Improving generalization for neural adaptive video streaming via meta reinforcement learning," in *Proc. 30th ACM Int. Conf. Multimedia*, Oct. 2022, pp. 3006–3016.

[17] S. Sengupta, N. Ganguly, S. Chakraborty, and P. De, "HotDASH: Hotspot aware adaptive video streaming using deep reinforcement learning," in *Proc. IEEE 26th Int. Conf. Netw. Protocols (ICNP)*, Sep. 2018, pp. 165–175.

[18] T. Huang, C. Zhou, R.-X. Zhang, C. Wu, X. Yao, and L. Sun, "Comyco: Quality-aware adaptive video streaming via imitation learning," in *Proc. 27th ACM Int. Conf. Multimedia*, Oct. 2019, pp. 429–437.

[19] *YouTube*. Accessed: Dec. 4, 2024. [Online]. Available: https://www.youtube.com

[20] *Netflix*. Accessed: Dec. 4, 2024. [Online]. Available: https://www.netflix.com

[21] *Bilibili*. Accessed: Dec. 4, 2024. [Online]. Available: https://www.bilibili.com

[22] X. Zhang, Y. Ou, S. Sen, and J. Jiang, "SENSEI: Aligning video streaming quality with dynamic user sensitivity," in *Proc. 18th USENIX Symp. Networked Syst. Design Implement. (NSDI 21)*, Apr. 2020, pp. 303–320.

[23] T. Badamdorj, M. Rochan, Y. Wang, and L. Cheng, "Joint visual and audio learning for video highlight detection," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2021, pp. 8107–8117.

[24] F.-T. Hong, X. Huang, W. Li, and W. Zheng, "MINI-Net: Multiple instance ranking network for video highlight detection," in *Proc. 16th Eur. Conf. Comput. Vis.-ECCV*, Aug. 2020, pp. 345–360.

[25] R. Zhan et al., "Deconfounding duration bias in watch-time prediction for video recommendation," in *Proc. 28th ACM SIGKDD Conf. Knowl. Discovery Data Mining*, Aug. 2022, pp. 4472–4481.

[26] A. Hore and D. Ziou, "Image quality metrics: PSNR vs. SSIM," in *Proc. 20th Int. Conf. Pattern Recognit.*, Istanbul, Turkey, Aug. 2010, pp. 2366–2369.

[27] R. R. R. Rao et al., "Bitstream-based model standard for 4K/UHD: ITU-T P.1204.3—Model details, evaluation, analysis and open source implementation," in *Proc. 12th Int. Conf. Quality Multimedia Exper. (QoMEX)*, May 2020, pp. 1–6.

[28] R. Qian et al., "Streaming long video understanding with large language models," in *Proc. 38th Int. Conf. Neural Inf. Process. Syst.*, Dec. 2024, pp. 119336–119360.

[29] X. Tang, J. Qiu, L. Xie, Y. Tian, J. Jiao, and Q. Ye, "Adaptive keyframe sampling for long video understanding," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2025, pp. 29118–29128.

[30] B. He et al., "MA-LMM: Memory-augmented large multimodal model for long-term video understanding," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2024, pp. 13504–13514.

[31] *ChatGPT*. Accessed: Mar. 8, 2024. [Online]. Available: https://openai.com/blog/chatgpt

[32] *GPT4-Vision*. Accessed: Mar. 8, 2024. [Online]. Available: https://chat.openai.com/g/g-NHeYHA2ik-gpt-vision-builder

[33] H. Liu, C. Li, Q. Wu, and Y. J. Lee, "Visual instruction tuning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2023, pp. 34892–34916.

[34] Y. Li, C. Wang, and J. Jia, "LLaMA-VID: An image is worth 2 tokens in large language models," 2023, *arXiv:2311.17043*.

[35] Y. Li, B. Hu, X. Chen, L. Ma, Y. Xu, and M. Zhang, "LMEye: An interactive perception network for large language models," *IEEE Trans. Multimedia*, vol. 26, pp. 10952–10964, 2024.

[36] *OpenAI Price*. Accessed: Mar. 8, 2024. [Online]. Available: https://openai.com/pricing

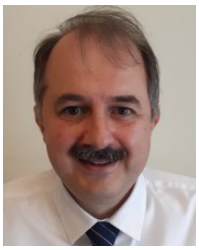
[37] A. Radford, J. W. Kim, T. Xu, G. Brockman, C. McLeavey, and I. Sutskever, "Robust speech recognition via large-scale weak supervision," in *Proc. Int. Conf. Mach. Learn.*, Jul. 2022, pp. 28492–28518.
[38] K. Lagler, M. Schindelegger, J. Böhm, H. Krásná, and T. Nilsson, "GPT2: Empirical slant delay model for radio space geodetic techniques," *Geophys. Res. Lett.*, vol. 40, no. 6, pp. 1069–1073, Mar. 2013.
[39] K. Zhou, Y. Qiao, and T. Xiang, "Deep reinforcement learning for unsupervised video summarization with diversity-representativeness reward," in *Proc. AAAI Conf. Artif. Intell.*, 2018, pp. 7582–7589.
[40] J. S. Park, J. O'Brien, C. J. Cai, M. R. Morris, P. Liang, and M. S. Bernstein, "Generative agents: Interactive simulacra of human behavior," in *Proc. 36th Annu. ACM Symp. User Interface Softw. Technol.*, Oct. 2023, pp. 1–22.
[41] A. Narayanan et al., "A variegated look at 5G in the wild: Performance, power, and QoE implications," in *Proc. ACM SIGCOMM Conf.*, Aug. 2021, pp. 610–625.
[42] M. Dasari, K. Kahatapitiya, S. R. Das, A. Balasubramanian, and D. Samaras, "Swift: Adaptive video streaming with layered neural codecs," in *Proc. 19th USENIX Symp. Networked Syst. Design Implement.*, Apr. 2022, pp. 103–118.
[43] Y. Liu, B. Jiang, T. Guo, R. K. Sitaraman, D. Towsley, and X. Wang, "Grad: Learning for overhead-aware adaptive video streaming with scalable video coding," in *Proc. 28th ACM Int. Conf. Multimedia*, Oct. 2020, pp. 349–357.
[44] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi, "Optimization by simulated annealing," *J. Stat. Phys.*, vol. 34, no. 5, pp. 975–986, 1983.
[45] Z. Du et al., "GLM: General language model pretraining with autoregressive blank infilling," in *Proc. 60th Annu. Meeting Assoc. Comput. Linguistics*, May 2022, pp. 320–335.
[46] M. Maaz, H. Rasheed, S. Khan, and F. S. Khan, "Video-ChatGPT: Towards detailed video understanding via large vision and language models," 2023, *arXiv:2306.05424*.
[47] R. Netravali, A. Sivaraman, K. Winstein, S. Das, A. Goyal, and H. Balakrishnan, "Mahimahi: A lightweight toolkit for reproducible Web measurement," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, pp. 129–130, 2014.
[48] H. Riiser, P. Vigmostad, C. Griwodz, and P. Halvorsen, "Commute path bandwidth traces from 3G networks: Analysis and applications," in *Proc. 4th ACM Multimedia Syst. Conf.*, Feb. 2013, pp. 114–118.
[49] Z. Akhtar et al., "Oboe: Auto-tuning video ABR algorithms to network conditions," in *Proc. Conf. ACM Special Interest Group Data Commun.*, Aug. 2018, pp. 44–58.
[50] D. Raca, J. J. Quinlan, A. H. Zahran, and C. J. Sreenan, "Beyond throughput: A 4G LTE dataset with channel and context metrics," in *Proc. 9th ACM Multimedia Syst. Conf.*, Amsterdam, Netherlands, Jun. 2018, pp. 460–465.

Hanling Wang received the B.S. degree from Central South University, China, in 2017, and the M.S. and Ph.D. degrees from Tsinghua University, China, in 2020 and 2025, respectively. He is currently an Assistant Researcher at the Peng Cheng Laboratory, China. His main research interests include video transmission and analytics, edge computing, and deep learning.

Tianli Zhou received the master's degree from Tsinghua Shenzhen International Graduate School, Shenzhen, China, in 2024. He is currently working with the AI Business Department, Alibaba International, focusing on large language model inference acceleration. His research interests include video transmission, video analysis using multimodal models, and AI infrastructure.

Qing Li (Senior Member, IEEE) received the B.S. degree in computer science and technology from Dalian University of Technology, Dalian, China, in 2008, and the Ph.D. degree in computer science and technology from Tsinghua University, Beijing, China, in 2013. He is currently a Full Professor at the Peng Cheng Laboratory, Shenzhen, China. His research interests include reliable and scalable routing of the internet, software-defined networking, network function virtualization, in-network caching/computing, edge computing, traffic scheduling, transmission control, and video delivery.

Yong Jiang (Member, IEEE) received the B.S. and Ph.D. degrees from Tsinghua University, Beijing, China, in 1998 and 2002, respectively. He is currently a Full Professor with the Division of Information Science and Technology, Tsinghua Shenzhen International Graduate School, Shenzhen, China, and the Department of Mathematics and Theories, Peng Cheng Laboratory, Shenzhen. He mainly focuses on the future internet architecture, the Internet of Things, edge computing, and AI for networks.



Gabriel-Miro Muntean (Fellow, IEEE) received the Ph.D. degree from Dublin City University (DCU), Ireland, in 2004, for research on adaptive multimedia transmission. He is currently a Professor with the School of Electronic Engineering and the Co-Director of the Performance Engineering Laboratory, DCU. He has authored over 500 papers in prestigious international journals and conferences, written four books and 29 book chapters, and edited six additional volumes. He managed the EU Project NEWTON and directed the DCU Team in the EU projects TRACTION and HEAT, as well as other significant Irish research projects, such as eStream and FRADIS. His research interests include quality, performance, and energy efficiency concerns associated with multimedia and multisensory media delivery, technology-enhanced learning, and various data transfers across heterogeneous networks. He was the chair, a reviewer, and an assessor of other prominent international conferences, journals, and funding organizations. He serves as an Associate Editor for IEEE TRANSACTIONS ON BROADCASTING and IEEE TRANSACTIONS ON NETWORK SCIENCE AND ENGINEERING. He is the Multimedia Communications Area Editor of IEEE COMMUNICATIONS SURVEYS AND TUTORIALS.