Unveiling Real-time Stalling Detection for Video Streaming Traffic

Ximin Li, Xiaodong Xu, Guo Wei, Xiaowei Qin

***Abstract*—In the rapidly evolving field of video traffic, ensuring a smooth video streaming experience for users is critical for network operators. Accurately and promptly detecting stalling events, a significant indicator of poor quality of experience, remains challenging due to varying detection time resolutions in existing techniques, which often detect stalls every video chunk, or every five or ten seconds. This paper makes three key contributions. First, we introduce the concept of detection granularities to enable fair performance comparisons and reveal their impact on detection performance from the data sampling perspective. Second, we propose a novel feature extraction approach that captures both packet-level and chunk-level features in a unified sequential manner to effectively detect stalling events. Third, a novel sample reweighting method is proposed to address the detection timeliness problem by focusing more on difficult samples around stalling starting or ending. Experimental results on both video-on-demand and live streaming traces demonstrate that our feature extraction approach achieves an average improvement of 5.3% in f1-score, 4.7% in coverage rate, and reduces stalling response time by 0.4 seconds compared to existing techniques. Additionally, the sample reweighting method further improves the detection sensitivity without compromising f1-scores for all detection techniques.**

***Index Terms*—Stalling detection, real-time monitoring, HTTP adaptive streaming, machine learning, quality of experience**

I. Introduction

VIDEO streaming has become predominant in Internet traffic in recent years, especially on mobile traffic. According to the 2022 Ericsson Mobility Report, video traffic constitutes 69% of all mobile data traffic, projected to increase to 79% by 2027 [1].

From the view of network operators, monitoring and improving user video quality of experience (QoE) is helpful to ensure user engagement and avoid user churn. Network operators generally implement traffic monitoring and adaptive network managements to address the continuously changing network conditions [2], [3]. Nevertheless, with the trend of end-to-end encryption of video traffic, traditional deep packet inspection techniques [4], [5] fail to extract video session information from HTTP sessions to infer video QoE metrics. Recent studies have turned to developing machine learning (ML) models to classify Key QoE Indicators (KQIs) based on encrypted traffic, such as stalls [6], [7], video quality levels [8], [9] and startup delay [10]. Among these, stalling is considered to have the most significant impact on QoE.

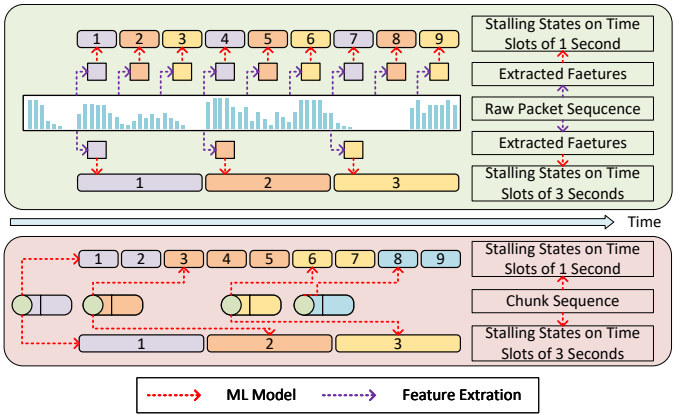


Fig. 1. Two paradigms exist for real-time stalling detection: detection at each time slot (upper) and detection at each chunk (lower), both leveraging encrypted video traffic and machine learning models. For the former, network features are extracted for each time slot to predict the stalling state within that slot. In contrast, the latter relies solely on historical chunk sequences, and the predicted state remains unchanged until a new chunk arrives.

Real-time stalling monitoring aims to detect stalling events as quickly as possible when they occur. Untimely detection of stalling events can hinder operators' ability to implement timely network management strategies and may potentially lead to a degraded user viewing experience. There are two main types of real-time detection paradigms: detect per chunk and detect per time slot, as shown in Figure 1. The former detects every chunk for lightweight inference, which usually employs a coarse-granted time resolution of chunk inter-request time [7], [11]. However, video chunks are not always arriving uniformly over time which depends on different buffer phases [12]. A large chunk inter-request time during the stalling period may compromise the timeliness of monitoring. The latter falls under fine-grained monitoring approaches. They generally leverage a set of equal features to handle multiple KQI tasks with ML models so that the task-related features may not be fully explored. They also ignore the temporal dependency of stalling state labels since training samples fed into the ML model are thought temporal-independent. In fact, the playback buffer mechanism ensures that stalling states exhibit infrequent changes, leading to a smooth stalling label sequence temporally. These methods also share a limitation: they do not evaluate the *timely* detection of stalls. Their performance is assessed primarily through a binary classification of the stalling state within each time slot,

This work was supported in part by the National Key Research and Development Program of China (2018YFA0701603), and the Natural Science Foundation of Anhui Province (2008085MF213). (*Corresponding author: Xiaowei Qin.*)

The authors are with the CAS Key Laboratory of Wireless-Optical Communications, University of Science and Technology of China, Hefei, Anhui 230026, China (e-mail: lxm123@mail.ustc.edu.cn; xdxu@ustc.edu.cn; wei@ustc.edu.cn; qinxw@ustc.edu.cn).

neglecting the latency aspect. In addition, whether coarse-grained or fine-grained, the frequency of detecting stalls is often determined by the time resolution of feature extraction, while the impact of this resolution on detection performance has not been fully explored.

In this work, our goals are 1) to investigate the impact of time resolutions on real-time stalling detection performance, and give a fair comparison between existing detection techniques with different resolutions, and 2) to provide a stalling-related feature extraction method that is able to minimizing manual feature engineering while effectively capturing the inherent characteristics of video streaming and playback behavior, 3) to design metrics to evaluate the timeliness of stalling detection, and to develop the model that ensures the detection timeliness as high as possible while maintaining stalling classification performance.

In sum, our contributions are as follows:

- Giving a comprehensive analysis of the impact of different time resolutions on detection performance. By introducing the concept of detection granularity, we can separate the time resolution of feature extraction (training granularity) from detection resolution (inference granularity) and evaluate their impact on detection performance respectively. We provide a detection benchmark on four video traces of both VoD and live streaming services, and fairly evaluate existing methods at any time resolutions. In addition, detection at each time slot and per chunk can be unified within this framework, treating them as uniform sampling and non-uniform sampling along the time dimension, respectively. The experiment results show the former can achieve more efficient detection performance under the same sampling ratio. These findings provide valuable guidance for parameter selection when deploying detection models, enabling informed granularity selection decisions to optimize performance and resource utilization. While previous work may have touched upon the concept of detection granularity, our work is the first to provide a clear definition and quantitatively analyze its impact on performance.
- Proposing a unified sequence-based feature extraction method that use the mode of *basic feature + long sequence* for feature construction. For each element in the sequence, only basic features are extracted to avoid complex statistics, and the feature hierarchy is enriched by controlling the sequence length. This method is universal regardless of packet-level and chunk-level features, or time-window and chunk sequences. Using only sequence data without explicitly modeling the temporal dependencies between sequence elements is sufficient for the ML model to achieve accurate classification.
- Introducing the sensitive metrics to evaluate model's detection timeliness. Analyzing the relationships between detecting stalling events and predicting stalling states by modelling the estimation error of the starts and ends of stalls. Following this hint, we present a distance-based reweighting method (DBR) that pays more attention on difficult samples around the time when the stall starts or ends, enhancing the model sensitivity on stalling detection without compromising classification performance.

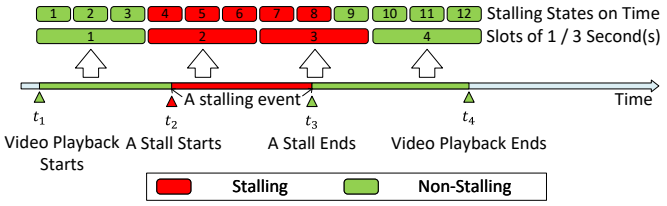


Fig. 2. The process of label transformation: A video plays from time t_1 to t_4, during which a stalling event occurs between t_2 and t_3. This corresponds to the stalling states in the time slot sequences as follows: the 4th to 8th time slots (for one-second slots), and the 2nd to 3rd time slots (for three-second slots), are labeled as stalling. All other slots are labeled as non-stalling.

II. Background and Related Work

This section introduces the background information of the VoD and live streaming delivery mechanisms and related works in the field of real-time stalling detection techniques.

A. Background of Adaptive Bitrate Streaming

HTTP Adaptive Streaming (HAS) has been widely applied to internet video transmission, whether they are VoD or live streamings. Videos are usually divided into variable-length video chunks, with clients requesting chunks from the server as the fundamental unit. The duration of a chunk often depends on the latency requirements of the service. Madanapalli et al. [13] revealed that VoD services, such as YouTube, typically employ chunks with a duration of 5 or 10 seconds, whereas live streamings offer a range of chunk durations, such as 1s, 2s, and 5s, depending on the latency mode. Each chunk may only contain a video segment or an audio segment or a mixture of the two. The audio is generally encoded at a constant bitrate, while video chunks are encoded using Adaptive Bitrate (ABR) to create chunks of varying quality levels.

When a chunk is downloaded, the playback will place this content into the buffer. The health of the playback buffer and its changing trend reflects different video states. Gutterman et al. [8] categorized the buffer state into buffer increase, buffer decay, steady, and stall, where a stalling event occurs when the remaining size of the buffer falls below a threshold. Stalls severely affect user's QoE, and accurate and timely detection of stalling events is helpful for Internet Service Provider (ISP) network management operations, which can benefit from real-time stalling detection methods.

B. Problem Statement and Chunk Separation Techniques

Problem Statement: Real-time stalling detection aims to enable ISPs to monitor encrypted traffic and detect the stalling events as quickly and accurately as possible. As illustrated in Figure 1, existing detection approaches typically transform the detection task into the time-slot-based stalling state prediction task, and the process of label transformation is illustrated in Figure 2. Using machine learning models, the stalling state (**stalling** or **non-stalling**) of each time slot is predicted. If multiple consecutive time slots are identified as being in a stalling state, the time period covering these slots is considered

This article has been accepted for publication in IEEE Transactions on Network and Service Management. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2025.3554822

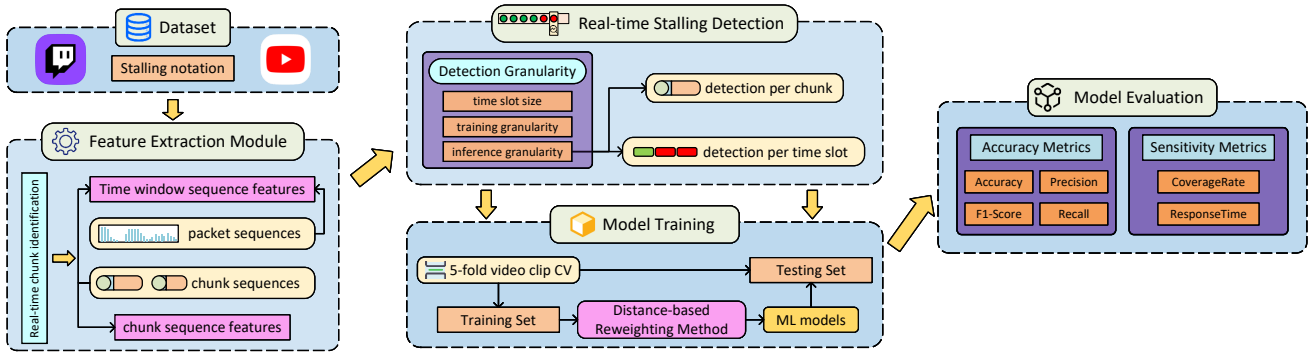


Fig. 3. The framework of stalling detection pipeline. Our contributions are threefold. Feature Extraction Module: we introduce sequence-based features that unify chunk-level and packet-level features as well as time-window and chunk sequences, enabling a consistent approach to feature representation. Detection Granularity Concept: We propose the concept of detection granularity, which provides a unified view for two predictive paradigms, which are detection per time slot and detection per chunk. Sensitivity metrics and DBR method: we introduce sensitivity metrics to evaluate the timeliness of stalling detection. Additionally, we proposed the DBR method, which enhances detection timeliness without sacrificing accuracy.

to correspond to a stalling event. Historical traffic features (from network layer, transport layer and application layer) are leveraged to train ML models and predict the stall state at each time slot. For example, given a video session, the stalling state y_i at the i-th time slot can be estimated by the traffic features $x_{t:t<t_i}$ before the i-th time slot, that is

$$\hat{y}_i = f(x_{t:t<t_i}) \tag{1}$$

where $f(\cdot)$ is the mapping of the ML classifier and t_i is the time of the i-th time slot.

Chunk Separation Techniques:Regarding historical traffic features, many studies employ chunk information from application layer, where individual chunks are first identified from encrypted traffic, and then their video state is predicted with chunk-based features. Furthermore, audio and video chunks are separated for fine-grained modeling, where the separation evidence can be request header lengths [8], [14], chunk download sizes [15], [16], chunk download durations [15], and flow ports [14]. However, these approaches either utilize the entire session's chunks [14] or the service-specific threshold to separate chunks [8], [13], [15]. The former contradicts the real-time requirement of chunk separation, while the latter limits the generalization of chunk features across different services. In this work, we extract chunk-level features in real-time, which does not involve video and audio separation, similar to the approach in [10].

C. ML-based Real-time Stalling Detection Techniques

In recent years, real-time QoE monitoring has witnessed significant advancements, particularly in ML-based approaches, which have gained popularity in both VoDs ([8], [9], [12], [17]–[19]) and live streamings ([13], [20]–[25], [25], [26]) domains. Although many studies claim to achieve real-time monitoring, the varying monitoring time resolutions lead to misalignment in evaluation metrics, making comparisons across different studies challenging.

In terms of real-time stalling detection, based on the layer from which features are extracted, methods can be categorized as packet-based methods from the network- and transport-layer [9], [18], [27] or chunk-based methods from the application-layer [8], [13], [14], [19], or a mixture of both [10], [17]. Generally, packet-based methods enable detection at a uniform time interval, such as detecting every time slot or time window, while chunk-level methods typically perform detection per each chunk. ViCrypt [9] achieved one-second-level detection by extracting network and transport layer packet-level features from three time window segments and achieved the f1-score of 68% with a Random Forest (RF) model. Shen et al. [27] utilized round-trip time features for detection but focused more on QoE estimation at the session level. Requet [8] suggested that modeling chunks can effectively represent the video transmission process, which employed various time windows to calculate statistics such as the number of chunks, average chunk size, and chunk download time. Loh et al. [19] performed chunk-level detection by sampling the fixed-length chunk sequence, where the performance of RF and long short-term memory models are compared, achieving f1-scores of 74% and 75%, respectively. Subsequently, Loh et al. [14] observed differences in the distribution of chunk inter-request time (IRT) across different video states, achieving a maximum recall of 92% by detecting stalls using only an IRT threshold. Madanapalli et al. [13] developed a buffer model to estimate the remaining buffer of live streaming in real time. Stalls can be reported when the buffer health fell below a predefined threshold, which enabled stalling detection at one-second-level.

In this work, we construct features from both time-window and chunk sequences, where packet-based and chunk-based features are both captured to enrich the feature representation. In addition, we also introduce the detection time granularity concept to achieve the alignment of detection per time slot or chunk, so that methods of different detection granularity can be compared under the same setting. The global framework of stalling detection pipeline is shown in Figure 3.

III. Dataset and Feature Construction

This section introduces the trace datasets of VoD and live streaming services used for stalling detection. We then present the real-time chunk identification approach and the feature

TABLE I
SUMMARY OF FOUR TRACES AFTER STALLING NOTATION.

Trace	Provider	Type	Protocol	Session Number	Total Duration (secs)	Stall Ratio
Trace A	YouTube	VoD	HTTPS + QUIC	6,391	1,845,877	22.0%
Trace B	YouTube	VoD	HTTPS + QUIC	215	128,690	22.4%
Trace C	YouTube	VoD	HTTPS	3,333	2,055,605	37.1%
Trace D	Twitch	Live	QUIC	6,885	3,830,334	32.0%

extraction module implemented by two sequence types, time-window and chunk sequences.

A. Trace Datasets

This work covers four video trace datasets, three traces from VoD and the other one from live streaming.

Trace A: Trace A is a public dataset [28] that includes 1,081 hours of video measurements captured across network, transport, and application layers using the native YouTube streaming client on mobile devices. The dataset contains 80 different network scenarios with 171 individual bandwidth settings across 11,142 streaming sessions from 246 different videos clips. It provides one-second-level quality indicators such as initial playback delay, streaming video quality, adaptive video quality changes, video stalling events, and streaming phases.

Trace B: Trace B is a public dataset from Requet [8], which collected a total of 425 sessions from 40 different video clips under the Browser-WiFi setting and App-LTE setting. Traces under the Browser-WiFi setting were collected in a laboratory environment with a laptop connected to WiFi networks, while others under the App-LTE setting involves native YouTube video sessions with the Android smartphone connected to LTE cellular network. It also provides video quality indicators at a time granularity of 0.1 second.

Trace C: Trace C is our YouTube dataset collected from 2022 Nov to 2023 May. We implemented an IFrame-based YouTube player on Android smartphones and collected video sessions under WiFi environments where the video playback can be recorded within a one-second granularity. 45 bandwidth scenarios similar to [28] were employed for the network diverse. The trace contains 3333 video sessions from 40 different video clips. It provides each video session's raw traffic pcap file captured by Tcpdump and one-second-level video KQI such as stalling, video resolution, initial delay and playback buffer information.

Trace D: Trace D is a live streaming dataset [26] collected by monitoring over 1,000 hours of Twitch live content across 6,982 streaming sessions from 222 different streamers. The dataset investigates 35 bandwidth scenarios, and each streaming session provides uplink request level or chunk-level playback logs. Each chunk contains request timestamp, download ending timestamp, chunk resolution, buffer health, latency, and various other player information.

Stalling Notation: For these traces, we determine different methods to label stalling events according to the streaming service. For VoD datasets, there are label imbalance issues that the stalling events rarely occur in the time dimension. According to the streaming phases in Trace A, there are only 2.3% time slots are stalling state and 78% of the sessions are without stalling time slots, leading to difficulty of performance evaluate and comparison. Therefore, we consider the buffer warning state, where the buffer level below 20 seconds as the stalling state of VoD like [8]. In addition, the buffer increase state at the start of playback does not include the stall to distinguish it from the initial delay. After labeling stalls, we remove the video sessions whose stalling ratio under 5% to balance the overall trace stalling ratio. Similar operations are also applied to Trace B and C. For live streaming, we follow the rule that video chunks with *minBuffer* below 2 seconds are stalling chunks as [26], while other chunks are non-stalling. For each session, we convert the buffer state at chunk level to the stalling state at time-slot level. We generate stalling states for each time slot by linearly interpolating the buffer health attribute. In this context, the stalling state refers to the actual occurrence of video playback interruptions due to insufficient buffer. The details of all traces are presented in Table I.

B. Real-time Chunk Identification

To implement a real-time stalling detection, we need firstly extract real-time features for each time slot. An important factor is the need to identify chunks in real time to ensure features constructed in a timely manner. To ensure the real time manner, we extract features from a mixed sequence containing both video and audio chunks without performing audio-video chunk separation so that we only focus on chunk identification in real time.

A complete chunk is detected by the start of an uplink request and ends with the detection of another chunk starting or no downlink packets received within one second. We identify the HTTP GET request by the IP payload length of the uplink packet using a threshold of 400B. An uplink packet with a payload length over the threshold is identified as a chunk request. By following this rule, the network trace from the same flow can be divided into the chunk sequence. We then combine sequences from different flows to construct a combined chunk sequence.

As shown in Figure 4, five chunk characteristics are recorded for each identified chunk, including the *request time* (the timestamp of sending the request for the chunk), the *request size* (the size of the request for the chunk), the *download start time* (the timestamp of receiving the first downlink packet for the chunk), the *download end time* (the timestamp of receiving the last downlink packet for the chunk),

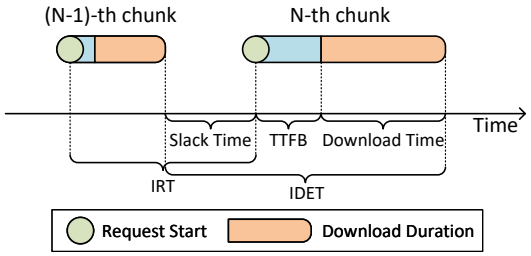


Fig. 4. Definition of chunk characteristics. For each chunk, we record three timestamps: the request sending time, the start time of receiving the chunk data, and the end time of receiving the chunk data. Based on these timestamps, we can calculate these characteristics, as the representation of the chunk.

and the *chunk size* (the amount of received data for the chunk). Furthermore, we merge all identified chunks into one sequence and compute the inter-request time (*IRT*) and inter-download-end time (*IDET*) between chunks in ascending order of the request start time and the download end time, respectively.

C. Feature Extraction Module

Following the real-time acquisition of chunk information, we proceed to introduce our feature extraction module. In contrast to previous studies that derive statistical distributions of various parameters with different time windows [9], [10], [18], we propose a unified approach to characterize packet-level and chunk-level features. Two distinct sequence types (time-window sequences and chunk sequences) are utilized to capture diverse traffic patterns within each time slot.

Time-Window Sequence Features. We introduce time-window sequences to capture the temporal patterns during the video playback. Unlike utilizing various different window sizes till the current time slot to represent short-term and long-term video patterns, we construct a time window sequence of the fixed window size in this work. For the sequence, each time window shares equal window size of ten seconds to achieve the balance of the complex of window and the length of sequence, where the choices of window sizes are experimented in the Section VI-B. The windows adjacent to each other do not overlap so that a sequence of N time windows containing the past $N \times 10$ seconds traffic, and features are generated independently in every time window. According to traffic patterns in the time window, the time window sequence features (TWSF) can be further divided into features for packet sequences and chunk sequences as shown in Figure 5.

TWSF for **packet sequences** only consider packet-level features in the time window. For VoD traces, we count the total bytes and number of packets in uplink and downlink traffic for both TCP and UDP packets in the time window. Moreover, we also count the ratio of 100-ms slots without any traffic in the window. There are in sum 9 features for each time window.

TWSF for **chunk sequences** extract chunk-level features in the time window. For each time window, we count the number of chunks (only the chunk whose download end time in the time window), and calculate the mean of *chunk size*, *download time*, *IRT* and *IDET*. Besides, we also calculate the mean time interval of chunks' *request time* and *download end time* to the time window end time respectively. Therefore, there are seven features for each time window. The design of TWSF of chunk sequences is intended to transform a sequence of chunks that are unevenly reached in the temporal dimension into features that can be uniformly outputted in the temporal dimension.

For both categories of features mentioned above, we have adopted a maximum sequence length of $N = 30$, with the longest considering historical information up to the past 300 seconds.

Chunk Sequence Features. Apart from extracting time-window sequence features, we also utilize raw chunk sequences to construct features. At each time slot, we collect the preceding M chunks as the chunk sequence. Four features of *chunk size*, *download time*, *IRT* and *IDET* characterize a chunk. The shortcoming of extracting features at chunk-level is that the chunk sequence is only updated when a new chunk arrives, thus the chunk sequence features remain unchanged until next chunk arriving. To differentiate the time slots when no new chunk arrives, we also introduce two new features to characterize a chunk, which are the time intervals of the current time slot to the *request time* and *download end time* of the chunk, respectively. Thus, each chunk is characterized by six features, and the entire chunk sequence consists of M chunks, ultimately resulting in $M \times 6$ chunk sequence features (CSF). For all four traces, we both set the maximum sequence length of $M = 60$ to capture long chunk dynamics patterns.

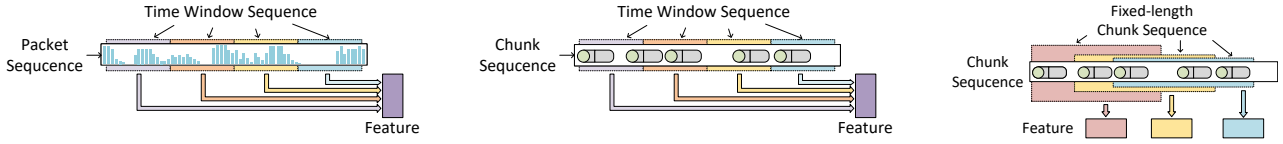
IV. ML-based Stalling Detection Modeling

In this section, we first introduce the definition of real-time detection granularity and analyze its impact on stalling detection performance. We then introduce the ML model and evaluation metrics utilized on the training and evaluation phase.

A. Real-time Detection Granularity

Fairly comparing the detection performance across various methods remains challenging due to the lack of a standardized trace collection pipeline and differences in time resolution during evaluation. Although many approaches claim real-time detection capabilities, variations in prediction time resolution hinder direct comparisons. For example, some methods provide second-level stalling detection [9], while others adopt time-window-level (detecting every five or ten seconds) or chunk-level approaches [8], [17], [19].

Typically, for each time slot, a feature sample is generated and fed into ML models to determine its stalling state. Different time resolutions imply that methods adopt varying time slot sizes, leading to differences in the sampling frequency of feature extraction during the ML model's training and evaluation phases. Our goal is to enable a fair comparison of detection performance under equal time slot sizes. To achieve this, we introduce the concepts of training granularity and inference granularity, decoupling the sample sampling frequency from the time slot size. This also allows us to investigate the impact of different time slot sizes under the



(a) TWSF for packet sequences (b) TWSF for chunk sequences (c) Chunk sequence features

Fig. 5. Two sequence types are leveraged to capture traffic features in this work. (a) and (b) time-window sequence features: we construct time-window sequences to extract features, collecting both packet-level and chunk-level features within each window. The features from each window are then organized into a time series, which serves as the final feature set. (c) chunk sequence features: We collect the fixed-length chunk sequence, extract corresponding features for each chunk, and combined the entire sequence into the final feature set.

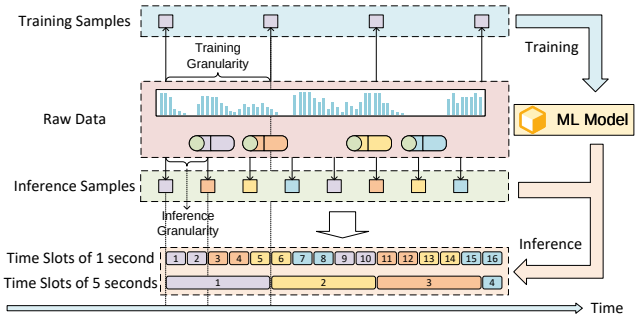


Fig. 6. A schematic diagram of detection granularities with different time resolutions. Training samples and inference samples are extracted at regular intervals using training granularity (e.g., 5 seconds) and inference granularity (e.g., 2 seconds), respectively. The former is used for training the ML model, while the latter is fed into the trained model to generate the corresponding stalling states. After the stalling state for each inference sample is predicted, it is mapped to the corresponding time slot based on predefined rules, thereby serving as the predicted stalling state for each time slot.

same sampling frequency. As shown in Figure 6, the following definitions are provided:

Time slot size. Time slot size represents the temporal resolution of ground-truth labels in the stalling state binary classification task, equivalent to sampling the stalling state from a continuous time sequence at fixed intervals as shown in Figure 2. In this work, the default time slot size is **one second**, meaning that the stalling state is sampled and annotated every second.

Training granularity. This defines the time interval for generating samples with feature extraction during the training phase. In other words, the training granularity determines the sampling frequency of input samples in the training dataset. For samples utilizing time-window-based features, the generation interval can be arbitrary, as the window can slide along the time dimension with sufficiently small time intervals. A larger training granularity compared to the time slot size results in undersampling of the training samples relative to the original training phase. If the training granularity is fixed, this corresponds to uniform undersampling along the time dimension, with the sampling ratio determined by dividing the time slot size by the training granularity. Conversely, a smaller training granularity than the time slot results in oversampling of the training samples compared to the original training phase. For samples derived from chunk-based features, the training granularity is not fixed but rather depends on the chunk arrival intervals, leading to non-uniform sampling across the time dimension.

Inference granularity. This describes the time interval at which the model outputs stalling state predictions with inference samples during the evaluation phase. Similarly to the training granularity, the inference granularity determines the sampling frequency at the evaluation phase. Reflected in the prediction of each time slot, the stalling state of a time slot is determined by adopting the output result of its most recent sample output.

Previous studies typically maintain equal inference and training granularities of the time slot size to ensure consistency in data distribution between training and evaluation phases. However, this makes direct comparisons between methods with different time slot sizes for different sampling. In this work, we split training and inference granularities to investigate their individual impacts on detection performance and offer deeper insights into the role of temporal granularity in real-time stalling detection. The related experiments are presented in Section VI-C.

B. ML Models

Tree-based models, such as RF and XGBoost [29], have gained recognition for their robustness and excellent performance in KQI monitoring tasks, especially when handling complex, high-dimensional data. This effectiveness is demonstrated by their consistently high f1-scores reported in various studies. Furthermore, given the features extracted from different sequences, neural networks designed for sequential data, such as recurrent neural networks and Transformers [30], can also be considered. In our preliminary experiments, we find that both tree-based models and sequence neural networks can achieve close detection performance, while the training and inference costs of neural networks are significantly higher than that of tree-based models. For optimizing the trade-off between training cost and inference performance, we utilize XGBoost to develop ML models in this work.

C. Evaluation Metrics

At the model evaluation phase, a variety of metrics are introduced to conduct a comprehensive assessment of the model. We not only employ accuracy metrics to evaluate binary classification performance, but also propose sensitivity metrics to evaluate the model timeliness in detecting stalling events.

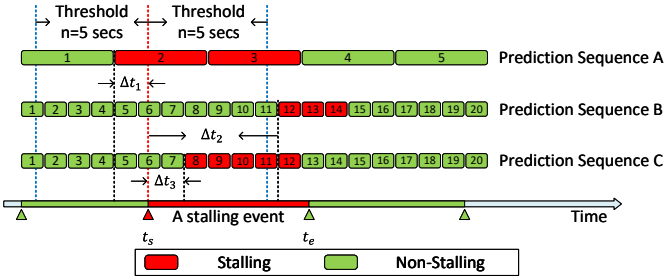


Fig. 7. An example of sensitivity metrics: Prediction Sequences A, B, and C represent the prediction results for different time slot sizes (4s, 1s, and 1s, respectively) in the stalling state classification task. A stalling event begins at time t_s and ends at time t_e. According to the definition, we can calculate the response time for detecting the start of the stalling event for each sequence ($\Delta t_1, \Delta t_2$ and Δt_3, respectively). Under the condition of a threshold $n = 5s$, Prediction Sequences A and C are considered successful detections, while Sequence B is deemed a failure.

1) Accuracy Metrics: For the evaluation, standard accuracy metrics of binary classification including *accuracy*, *precision*, *recall* and *f1-score* are utilized to evaluate prediction performance of the stalling state prediction task. Samples in video sessions are evaluated with the time resolution of the time slot.

2) Sensitivity Metrics: In addition to evaluating binary classification performance, we are also highly concerned with the timeliness of stalling event detection, which has not been adequately considered in previous work. In fact, due to the varying durations of stalling events, the timeliness of stalling detection cannot be effectively reflected solely by binary classification performance. Furthermore, we propose two metrics to characterize the model's sensitivity in detection stalling events, *CoverageRate@n* (CR@n) and *ResponseTime@n* (RT@n).

We denote $\tilde{\boldsymbol{y}}^i_{t_j}$ as the model prediction of the sample $\boldsymbol{x}^i_{t_j}$, where i refers to the i-th video session in the whole S sessions and t_j refers to the time of the j-th time slot. Similarly, the predicted time slot sets at stalls starting and ending can be summarized. These are $\tilde{\boldsymbol{T}}_s = \{(i,t_j)|\tilde{\boldsymbol{y}}^i_{t_j} = 1, \tilde{\boldsymbol{y}}^i_{t_{j-1}} = 0, i = 1,2,\ldots,S\}$ and $\tilde{\boldsymbol{T}}_e = \{(i,t_j)|\tilde{\boldsymbol{y}}^i_{t_j} = 0, \tilde{\boldsymbol{y}}^i_{t_{j-1}} = 1, i = 1,2,\ldots,S\}$, respectively. We can also obtain the ground-truth of time stalling events starting and ending from the raw trace collection, respectively, which are $\boldsymbol{T}_s = \{(i,t_s)|i = 1,2,\ldots,S\}$ and $\boldsymbol{T}_e = \{(i,t_e)|i = 1,2,\ldots,S\}$ where t_s respect the start of stalls and t_e respect the end of stalls.

CoverageRate@n reveals the model's hit rate in predicting stalls starting or ending within n seconds, indicating the model's ability to detect a stalling event within n seconds. For the time a stall starts, if the stalling states of the time slots within n seconds before and after that time are predicted accurately, we consider that the model has successfully detected the start of the stall. The same applies to the end of the stall. In this way, the hit rate of stalling starts and ends can be measured, which is referred to as CR@n. A model with higher CR@n presents that it is able to detect stalling events more timely. We define the indicator functions $I^n_s(i,t)$ and $I^n_e(i,t)$ to indicate whether the model hit the start and end of a stalling event:

$$I^n_s(i,t_s) = \begin{cases} 1, & \min\limits_{(i,t_j)\in\tilde{\boldsymbol{T}}_s} |t_j - t_s| \le n \\ 0, & \text{others} \end{cases} \tag{2}$$

$$I^n_e(i,t_e) = \begin{cases} 1, & \min\limits_{(i,t_j)\in\tilde{\boldsymbol{T}}_e} |t_j - t_e| \le n \\ 0, & \text{others} \end{cases}. \tag{3}$$

Therefore, the $CR@n$ can be expressed as:

$$CR@n = \frac{1}{|\boldsymbol{T}_s| + |\boldsymbol{T}_e|}\left(\sum_{(i,t_s)\in\boldsymbol{T}_s} I^n_s(i,t_s) + \sum_{(i,t_e)\in\boldsymbol{T}_e} I^n_e(i,t_e)\right). \tag{4}$$

ResponseTime@n shows the mean absolute estimation error between the actual start/end time of a stalling event and the predicted start/end time. Particularly, if the model fails to predict the start/end of the stall event within the time threshold of n seconds, we set the response time to n seconds. Therefore, the expression of RT@n is that:

$$RT@n = \frac{1}{|\boldsymbol{T}_s| + |\boldsymbol{T}_e|}\left(\sum_{(i,t_s)\in\boldsymbol{T}_s} \min\{n, \min_{(i,t_j)\in\tilde{\boldsymbol{T}}_s} |t_j - t_s|\} + \sum_{(i,t_e)\in\boldsymbol{T}_e} \min\{n, \min_{(i,t_j)\in\tilde{\boldsymbol{T}}_e} |t_j - t_e|\}\right). \tag{5}$$

A model with low response time indicates that it shows the ability of detecting a stall starts and ends accurately as fast as possible.

The sensitivity metrics are proposed to evaluate stalling detection from the perspective of identifying complete stalling events, whereas binary classification tasks primarily assess model performance globally. In comparison, longer stalling events contribute more to accuracy metrics, as they consist of more time slots labeled as stalling. However, shorter stalling events are underrepresented, even though, from the ISP's perspective, the timeliness of detecting each stalling event is equally important. To address this, our metrics treat each stalling event as the smallest unit for evaluation. For each event, we measure the model's response time. A stalling event is considered successfully detected only if its is below a predefined threshold; otherwise, it is deemed a failure. Thus, the coverage rate can be viewed as the hit rate or the recall rate of stalling event detection, while response time serves as a first-order statistic describing the detection delay distribution across all stalling events.

The setting of the threshold is intended to prevent the model from ignoring entire stalling events, which would result in an infinite detection delay that cannot be calculated. Therefore, an upper limit is set for the detection delay, and events with delays exceeding this limit are recorded as the upper threshold. This leads to the calculated $RT@n$ being smaller than the actual delay, especially when n is small, where the deviation becomes more pronounced. As a result, we typically choose a larger value for n to achieve a more accurate delay measurement. However, setting n too large reduces the relevance of $CR@n$, as stalling events with excessively long

delays often indicate that the user experience has already been compromised. Therefore, we aim to select a more balanced n value, such as 5 seconds or 10 seconds, to ensure the validity of both metrics.

V. Sample Reweighting Method

A. Estimation Errors of Detecting Stalling Events

As presented in Section II-B, ML models focus on modeling the stalling states at all time slots. We similarly introduce the task of predicting the event state of stalling events. The task aims to detect whether a stall starts or ends occurs in each time slot. We define the event state as the stalling state changes in the time dimension, and the event state of the i-th time slot, denoted as $z_i = y_{t_i} - y_{t_{i-1}}$. When z_i is 0, no stalling event occurs or ends, while occurs when z_i is 1 or -1, respectively. Thus, the goal of stalling event detection is to estimate the event state ($0, -1$ and 1) at each time slot directly. Compared to predicting stalling states, this task has a more imbalanced label distribution, but it also better matches the domain-specific knowledge that *stalling and non-stalling playback tend to occur in a continuous manner*.

To model the relationships between the stalling states and the event states, we first define the *switch distance* as the time interval of a time slot to its closest time of a stall starting or ending, which is

$$l(i, t_j) = \min_{(i,t) \in \boldsymbol{T}_s \cup \boldsymbol{T}_e} |t_j - t| \tag{6}$$

for the time of the j-th time slot at the i-th video session. Given an item (i, t) in the set $\boldsymbol{T}_s \cup \boldsymbol{T}_e$, we assume that an ideal ML model yields exactly one event state prediction (-1 or 1) corresponding to the item, with the time of the prediction time slot denoted as $\hat{t}$. The estimation error between the stalling event is $r(\hat{t}, t) = \hat{t} - t$, where existing works utilized the estimation error to evaluate the KQIs like the startup delay [9], [10], [19].

Supposing that the estimation error distribution of the stall occurrence time $p(r)$ follows a Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$, we consider the model prediction error rate distribution with different switch distances l for the stalling states classification task, delineated as $q(l; \mu, \sigma)$. For a stall starting at the time t, consider the j-th and k-th time slots where $m = t_j - t = t - t_k$ ($m > 0$). The stalling state is wrongly predicted only when the prediction error $r(\hat{t}, t)$ falls in the range $(m, +\infty)$. Similarly, the stalling state is wrongly predicted when $r(\hat{t}, t) \in (-\infty, -m)$. The samples with switch distance m are at the j-th and k-th time slots, so the prediction error rate at the switch distance m is

$$\begin{aligned} q(l = m; \mu, \sigma) &= \frac{1}{2}\left(\int_{m}^{+\infty} p(r)dr + \int_{-\infty}^{-m} p(r)dr\right) \\ &= \frac{1}{2}\left(1 - \Phi(\frac{m-\mu}{\sigma}) + \Phi(\frac{-m-\mu}{\sigma})\right) \end{aligned} \tag{7}$$

where $\Phi(x)$ denotes the cumulative function of the standard normal distribution. It is observed that the distribution is symmetric about $\mu = 0$, therefore we only consider the case where $\mu \geq 0$. Figure 8 demonstrates the impact of varying μ and σ values on the distribution $q(l; \mu, \sigma)$. Overall, regardless of the values of μ and σ, the error rate is always 50% at $l = 0$, and decreases to zero when l grows. Figure 9a and Figure 9b respectively show the curves of prediction error rates per switch distance and the probability density function (PDF) curves of samples over increasing switch distance across four traces under our selected features sets trained with the XGBoost model. As the figure presented, all traces show the same trend of prediction error rate with the switch distance that it is 50% at the zero switch distance and decreases as the distance increases.

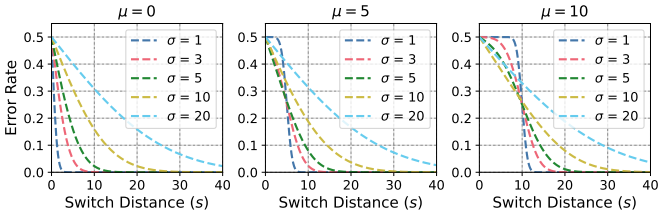
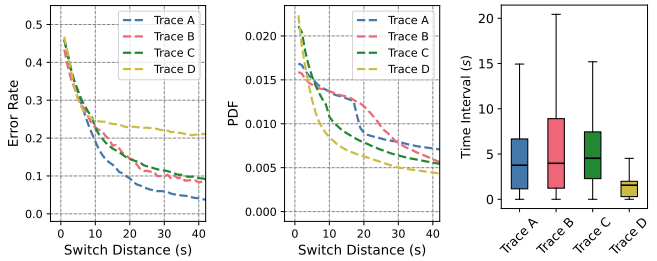


Fig. 8. Curves of prediction error rates against switch distances with varying parameters μ and σ.



(a) Error rate (b) PDF (c) IRT

Fig. 9. Comparative analysis of prediction XGBoost models across four traces. (a) prediction error rates against switch distances. (b) cumulative distribution functions of samples with switch distances. (c) the IRT distributions of four traces.

B. Distance-based Sample Reweighting Method

Based on the above observations, we can use switch distance as a difficulty metric for the stalling state classification task. The larger the switch distance, the lower the difficulty of the prediction. We aim for the model to accurately predict the occurrence of events for timely detection. Therefore, we propose a sample reweighting method to improve global prediction performance and timeliness. Weights are assigned to each training sample based on their switch distance, with larger distances resulting in smaller weights, which ensures that the model is able to effectively mine difficult samples. The original weight of 1.0 is assigned to the sample with the smallest distance of zero. The weights decrease exponentially as the distance increases, with the rate of decrease controlled by the parameter L. Additionally, to avoid assigning too small a weight to samples that are too far away, a fixed weight is assigned to samples whose weights less than γ. The weights for samples at different switch distances can be summarized as follows

$$\lambda(l) = \min\left(\exp\left\{-\frac{l}{L}\right\}, \gamma\right). \tag{8}$$

The distance-based reweighting method (DBR) can be applied to any ML-based model as well as any feature extraction approaches that support the ability to assign weights to samples.

VI. Performance Evaluation

In this section, we evaluate detection performance using different approaches and settings for four traces. First, the performance evaluation of different methods and feature sets from both aspects of accuracy and sensitivity is given. Next, we examine how the detection granularity affects the detection performance. Finally, the performance of DRM is also shown.

A. Experimental Setup

Data traces. The evaluation contains four traces introduced in Section III-A. For each dataset, the evaluation is done through the 5-fold stratified cross-validation where folds are split by video clips rather than by video sessions or individual time-slot samples. This ensures that samples from the same video clip are not present in both the training set and the validation set at the same time.

Training Setup. We implement the framework under the packages of scikit-learn [31] and XGBoost [29]. The XGBoost configuration is that the number of estimators is 500 and the other hyperparameters remain the default. All the experiments run on a desktop with Intel Xeon E5-2650 CPU and 64GB memory.

Comparative Experiment. We choose Orsolic's work [18], ViCrypt [9] and Requet [8] as methods for comparing VoD traces, Requet and Loh's work [26] for live streaming traces. For a fair comparison, only the feature extraction components of these methods are retained, and the classifiers are trained using XGBoost with the same configuration.

1) *Orsolic's work.* We utilize the 228 features for real-time KPI prediction from the traffic trace on window sizes of 1, 2, 3, 5, 10, and 20 seconds as the original work [18]. Since this work's time slot size is aligned with our default size of one second, no additional post-processing is performed on the detection output.
2) *ViCrypt.* We calculate the 208 features from the 1-second trend window, the 3-second trend window and the session window for each time-slot sample as ViCrypt [9]. Similar to Orsolic's approach, ViCrypt model has the time slot size of one second, ensuring all extracted samples are trained and evaluated.
3) *Requet.* The Requet system [8] extracts audio and video chunk features based on time windows of between 10 and 200 seconds. As the original work did not provide the exact threshold for identifying video and audio chunks based on HTTP request length, we implement a KMeans classifier for unsupervised clustering. In each session, we first extract all chunks as described in Section III-B, and record each chunk's request length and downlink size. Then, these two attributes are employed to train a KMeans classifier, dividing the chunks into two distinct classes. In one class, chunks with a higher average request length are classified as video chunks, while in the other class, chunks are classified as audio chunks. Moreover, time-window-based audio/video chunk features can be extracted separately. In fact, the use of a classifier to identify different types of chunks does not meet requirements for timely detection. Although the original work detected stalls every five seconds or per chunk, we still follow the one-second time slot for feature construction.
4) *Loh's work.* We employ the selected set with the best performance as the comparative method. In this feature set, the request size, port, downlink size and protocol are recorded for each chunk. The port and protocol are transformed to categorical features. Each training/inference sample is represented by its closest sequence of six chunks, resulting in a total of 24 features. The original work predicted the stall event for each chunk. Therefore, the chunk-level inference must be transformed into an inference for every time slot, as mentioned in Section IV-A.
5) *Our work.* A combination of time-window sequence features is employed to represent each time-slot sample, consisting of both packet sequences and chunk sequences, in addition to chunk sequence features. This generates the full feature set. The sequence lengths of TWSF and CSF are 30 and 60, respectively, with a window size of 10 seconds for TWSF. This results in a total of $(9+7)\times 30+6\times 60 = 810$ features per sample for VoD traces, and $7 \times 30 + 6 \times 60 = 570$ features for the Trace D, which lacks the raw packet sequences. Furthermore, the RFE-based feature selection method is employed to extract the 200 features with the highest feature importance from each trace, thus constituting the selected feature set.

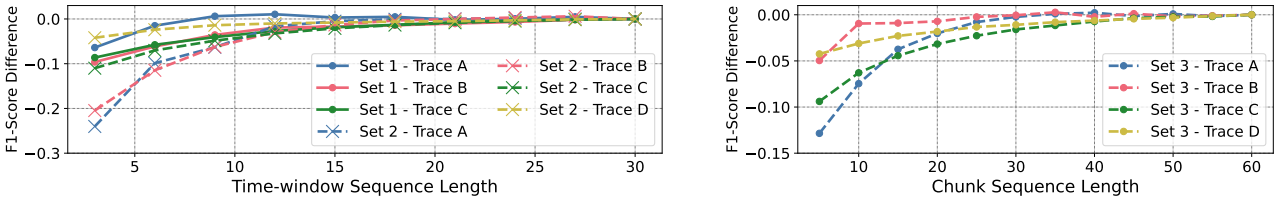
B. Experiment for the Detection Performance Benchmark

Performance Benchmark. We first verify the stalling detection abilities of the above five methods from both aspects of accuracy and sensitivity. The experimental results on the four traces are shown in Table II. The proposed method, utilizing both the full feature set and the selected feature set, demonstrates significantly superior performance compared to all other approaches in terms of accuracy and sensitivity metrics across all traces. This is evidenced by an average increase of 5.3% in f1-score, a 4.7% improvement in CR@10, and a reduction of 0.4 seconds in RT@10. Notably, the performance difference using the full feature set, and the selected features set is minimal, indicating the effectiveness of our approach even with a reduced feature subset. Moreover, Requet exhibits superior performance in comparison to remaining other methods in terms of all evaluated metrics. The result demonstrates that employing chunk-level features from a longer historical time window enhances the capability of detecting stalls, in contrast to the less effective approach of analyzing intricate packet patterns within shorter windows.

Feature Set Comparison. The full feature set is partitioned into three distinct subsets (Set 1, 2, and 3), each with specific characteristics. Set 1 represents the TWSF for packet

TABLE II
STALLING DETECTION PERFORMANCE OVER FIVE METHODS ACROSS THE FOUR TRACES.

Trace	Method	Accuracy	Stalling			Non-Stalling			CR@10	RT@10(s)
			Precision	Recall	F1-Score	Precision	Recall	F1-Score		
Trace A	Orsolic's work [18]	0.8529	0.5526	0.7137	0.6229	0.9375	0.8815	0.9086	0.6227	6.329
	ViCrypt [9]	0.8427	0.6141	0.6508	0.6319	0.9072	0.8930	0.9000	0.4027	7.710
	Requet [8]	0.8939	0.7205	0.7800	0.7491	0.9427	0.9229	0.9327	0.6024	6.475
	Our work (full set)	0.9162	0.7773	0.8308	0.8032	0.9554	0.9384	0.9468	0.6600	6.014
	Our work (selected set)	**0.9192**	**0.7844**	**0.8377**	**0.8102**	**0.9572**	**0.9403**	**0.9487**	**0.6695**	**5.981**
Trace B	Orsolic's work [18]	0.8278	0.5161	0.6443	0.5732	0.9178	0.8679	0.8922	0.5800	6.125
	ViCrypt [9]	0.8475	0.5633	0.6978	0.6233	0.9296	0.8806	0.9044	0.4542	7.071
	Requet [8]	0.8515	0.6062	0.6924	0.6464	0.9223	0.8903	0.9060	0.5113	6.826
	Our work (full set)	0.8837	0.7024	0.7601	0.7301	**0.9360**	0.9159	0.9259	0.6187	**5.818**
	Our work (selected set)	**0.8870**	**0.7177**	**0.7635**	**0.7399**	0.9358	**0.9199**	**0.9278**	**0.6227**	5.852
Trace C	Orsolic's work [18]	0.7698	0.7115	0.6817	0.6963	0.8041	0.8254	0.8146	0.5307	6.718
	ViCrypt [9]	0.8190	0.7585	0.7547	0.7566	0.8546	0.8572	0.8559	0.3672	7.725
	Requet [8]	0.8368	0.7974	0.7707	0.7838	0.8601	0.8780	0.8690	0.4995	6.823
	Our work (full set)	0.8605	0.8237	**0.8049**	0.8142	**0.8822**	0.8946	0.8884	0.5259	6.615
	Our work (selected set)	**0.8608**	**0.8253**	0.8046	**0.8148**	0.8818	**0.8954**	**0.8885**	**0.5334**	**6.577**
Trace D	Requet [8]	0.7638	0.5564	0.6536	0.6011	0.8613	0.8050	0.8322	0.3730	7.416
	Loh's work [26]	0.7079	0.4032	0.5604	0.4690	0.8513	0.7521	0.7986	0.3605	7.537
	Our work (full set)	**0.7772**	**0.6032**	0.6571	**0.6290**	0.8565	**0.8256**	**0.8407**	**0.4501**	**6.761**
	Our work (selected set)	0.7743	0.5938	**0.6648**	0.6273	**0.8592**	0.8181	0.8382	0.4423	6.825



(a) Set 1 and 2 (b) Set 3

Fig. 10. The f1-score gap compared to the original sequence length with varying sequence length across different feature sets.

TABLE III
STALLING DETECTION PERFORMANCE OVER DIFFERENT FEATURE SETS ACROSS FOUR TRACES.

Trace	Feature Set	Accuracy	F1-Score	CR@10	RT@10(s)
Trace A	Set 1	0.8786	0.7081	0.5660	6.498
	Set 2	0.9018	0.7659	**0.6523**	**5.756**
	Set 3	**0.9071**	**0.7809**	0.5326	6.994
Trace B	Set 1	0.8582	0.6509	0.5675	6.426
	Set 2	0.8628	0.6602	**0.7113**	**5.283**
	Set 3	**0.8793**	**0.7319**	0.5990	6.082
Trace C	Set 1	0.8294	0.7716	0.4801	7.055
	Set 2	0.8449	0.7924	**0.5173**	**6.633**
	Set 3	**0.8562**	**0.8088**	0.4816	6.930
Trace D	Set 2	**0.7707**	**0.6189**	0.3927	7.307
	Set 3	0.7645	0.6094	**0.4477**	**6.790**

TABLE IV
F1-SCORE OVER DIFFERENT WINDOW SIZES OF TWSF ACROSS FOUR TRACES.

Feature Set	Trace	Window Size(s)				
		5	10	20	40	50
Set 1	Trace A	**0.7173**	0.7052	0.6941	0.6572	0.6428
	Trace B	0.6285	0.6384	0.6446	**0.6511**	0.6439
	Trace C	0.7603	**0.7612**	0.7596	0.7486	0.7446
Set 2	Trace A	0.7558	**0.7629**	0.7628	0.7502	0.7547
	Trace B	0.6428	0.6618	0.6720	**0.6818**	0.6681
	Trace C	0.7754	0.7846	**0.7860**	0.7820	0.7781
	Trace D	**0.6141**	0.6131	0.6112	0.6084	0.6067

sequences, Set 2 encapsulates the TWSF for chunk sequences, and Set 3 represents the CSF. Table III illustrates the detection performance of different subsets across four traces.

Regarding VoD traces, Set 2 consistently outperforms Set 1 in terms of both accuracy and sensitivity. This suggests that, when the time-window sequence length is equal, chunk-based features are more effective at characterizing stalls compared to packet length features. While Set 3 demonstrates higher accuracy and f1-score than Set 2, it exhibits a lower coverage rate and higher response time. Conversely, in Trace D of live streaming videos, Set 2 achieves higher accuracy and f1-score, whereas Set 3 excels in sensitivity metrics. These performance discrepancies may be attributed to the differing distributions of video chunk request intervals between VoD and live streaming services. The longer time intervals between VoD chunk requests allow fixed-length chunk sequences to capture more historical data compared to live streaming scenarios.

Sequence Length and Window Size. The benchmark analysis reveals that features with a longer sequence are more effective in detection performance. We further investigate the impact of sequence length on performance. The f1-scores

TABLE V
STALLING DETECTION PERFORMANCE COMPARISON BETWEEN TWSF AND VARYING-WINDOW-SIZE FEATURES.

Trace	Feature Set	Accuracy	F1-Score	CR@10	RT@10(s)
Trace A	Set 1	**0.8786**	**0.7081**	**0.5660**	**6.498**
	Set 1s	0.8625	0.6698	0.4935	7.159
	Set 2	**0.9018**	0.7659	**0.6523**	**5.756**
	Set 2s	0.9015	**0.7664**	0.6376	6.041
Trace B	Set 1	0.8582	**0.6509**	**0.5675**	**6.426**
	Set 1s	**0.8589**	0.6501	0.5034	6.863
	Set 2	**0.8628**	**0.6602**	**0.7113**	**5.283**
	Set 2s	0.8596	0.6595	0.6483	5.771
Trace C	Set 1	**0.8294**	**0.7716**	**0.4801**	**7.055**
	Set 1s	0.8180	0.7586	0.4295	7.498
	Set 2	**0.8449**	**0.7924**	**0.5173**	**6.633**
	Set 2s	0.8368	0.7838	0.4995	6.822
Trace D	Set 2	**0.7707**	**0.6189**	**0.3927**	**7.307**
	Set 2s	0.7701	0.6182	0.3881	7.323

of the three feature subsets are recorded at time-window and chunk sequence lengths of 30 and 60, respectively. The sequence length is gradually reduced, and the resulting truncated feature subsets are employed for training and evaluation. This process allows us to quantify the difference in f1-score compared to the original sequence length. Figure 10 illustrates the change in the performance gap between each feature subset and the original feature set as the sequence length is reduced.

As depicted in Figure 10a, the f1-score exhibits a downward trend with decreasing time-window sequence length, except for Set 1 on Trace A. Furthermore, the rate of decline appears to accelerate as the sequence length becomes shorter. Notably, for any given sequence length, Set 2 demonstrates a greater decrease in f1-score compared to Set 1 across VoD traces. Utilizing only the 3 most recent time windows, representing the last 30 seconds of traffic data, results in a 5% to 22% reduction in f1-score on VoD traces compared to the origin sequence length. Conversely, Figure 10b reveals that reducing the chunk sequence length has a smaller impact on classification performance. The f1-score remains constant when the sequence length is varied between 30 and 60.

Additionally, we examine the influence of varying time-window sizes on both chunk-level and packet-level features. We maintain a constant total historical information length of 200 seconds. Various window sizes (5, 10, 20, 40, and 50 seconds) are selected, each associated with a distinct window sequence length, to extract features. Table IV presents the f1-scores of Set 1 and 2 with different window sizes across four traces. The experimental results demonstrate that the optimal window size varies across different traces for both Set 1 and Set 2.

TWSF versus Variable-Windows-Based Features. To evaluate the effectiveness of sequence-based features, we transformed our original feature Set 1 and 2, based on a fixed time window sequence, into feature sets based on varying window sizes. Specifically, starting with TWSF with a sequence length of 30 and an initial window size of 10 seconds, we generated new feature sets with window sizes of 10, 20, ..., 300 seconds, similar to the approach in Requet [8]. For each of these window sizes, we recalculated the features corresponding to the original Set 1 and Set 2, creating transformed feature sets denoted as Set 1s and Set 2s, respectively. We compared the performance of detection models using the original and transform feature sets on four network traffic traces.

As shown in Table V, the original sequence-based feature set outperformed the feature sets with varying window sizes in the majority of comparisons across all metrics. This demonstrates the superiority of the time-window sequence approach for feature construction.

C. Experiment for the Detection Granularity

We next investigate the impact of detection granularity variations on detection performance, considering three distinct scenarios: one-second granularity, five-second granularity, and chunk-level granularity.

One-second-level Detection. We begin by analyzing the default time-slot-level detection scheme, where the time slot size is one second. The performance trends under three distinct scenarios as the detection granularity changes are compared. In scenario 1, the training granularity is fixed at one second, while the inference granularity is adjusted. Scenario 2 involves fixing the inference granularity to 1 second and adjusting the training granularity. Finally, in scenario 3, both training and inference granularities are modified simultaneously while maintaining equal. The full feature sets on four traces are utilized, with granularities ranging from one to ten seconds.

Figure 11 presents the trends of f1-score, CR@10, and RT@10 metrics across these three scenarios as the granularity changes. The figure clearly demonstrates that an increase in inference granularity leads to a uniform decline in all three metrics for all traces when maintaining a fixed training granularity. For instance, in Trace A, a 10-second inference granularity results in a 1.5% reduction in f1-score, a 16.6% decrease in CR@10, and a 1.69-second increase in response time compared to the one-second-level inference. This observation aligns with the expectation that increasing the inference interval diminishes both sensitivity and accuracy of the detection process.

For scenario 2, the result reveals that with the increase of training granularity, the f1-score of the four traces also decreased, but the decrease is smaller than that of scenario 1 on VoD traces. A higher f1-score can be achieved on Trace A even when utilizing a training granularity from two to four seconds. Sampling 10% of the samples results in only average 0.76% f1-score decrease on four traces. In terms of CR@10 and RT@10 metrics, the performance remains constant as the training granularity increases. This result indicates that for the stalling detection task, fine-grained sampling in the training phase will lead to a certain degree of data redundancy. **Uniform undersampling of generated samples in the time dimension may result in a slight loss of accuracy, while the detection sensitivity does not change much.** Ultimately, the choice of training granularity depends on the trade-off between training cost and detection performance.

In scenario 3, where both training and inference granularities are increased simultaneously, Figure 11 demonstrates that the observed trends in the three metrics can be interpreted

This article has been accepted for publication in IEEE Transactions on Network and Service Management. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2025.3554822

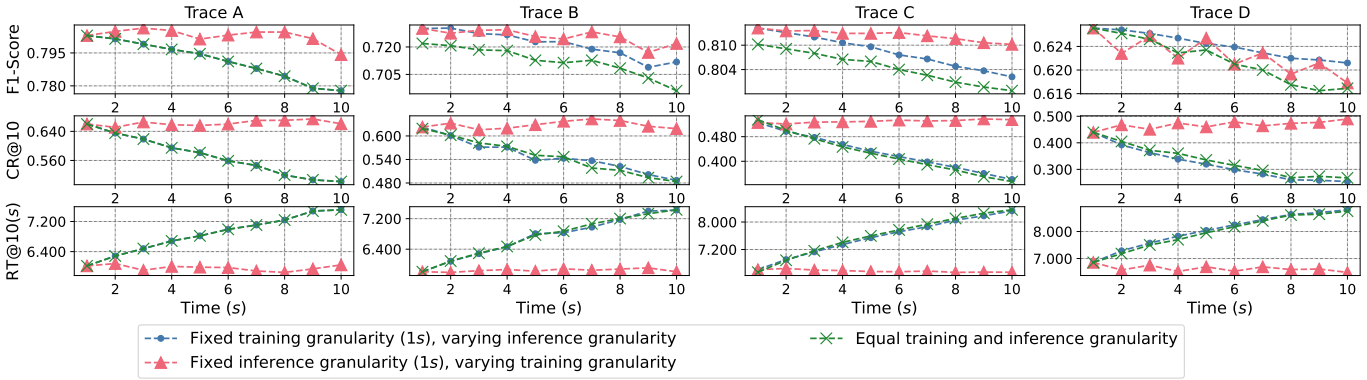


Fig. 11. The detection performance trends of the one-second-level detection with varying detection granularities on four traces. The three lines (blue, red, and green) in the figure correspond to three different scenarios (Scenario 1, 2, and 3). As shown, when the time slot is set to 1 second, increasing the training granularity alone does not significantly degrade detection performance. However, increasing the inference granularity leads to a sharp decline in performance, both in classification accuracy and detection timeliness. This could be attributed to the inherent inference delay introduced by larger inference granularities. Furthermore, increasing both granularities results in performance similar to the scenario where only the inference granularity is increased.

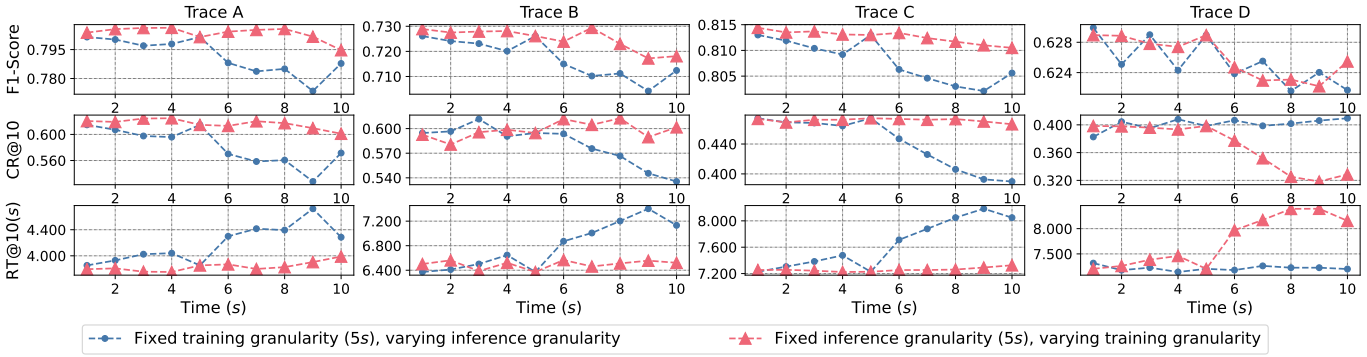


Fig. 12. The detection performance trends of the five-second-level detection with varying detection granularities on four traces. The two lines (blue and red) in the figure correspond to two different scenarios (Scenario 1 and 2). When either the training or inference granularity is below 5 seconds, the detection performance remains largely comparable to that observed at the 5-second granularity. However, when the granularity exceeds 5 seconds, the performance trend mirrors that observed at the one-second time slot level: increasing only the training granularity results in a slight performance drop, while increasing only the inference granularity causes a sharp decline in detection performance. Notably, this turning point aligns closely with the time slot size.

as a combination of the trends observed in the previous two scenarios. Specifically, scenario 3 exhibits the most rapid decline in f1-score among all three scenarios. In terms of CR@10 and RT@10, scenario 3 follows a similar trend to scenario 1. Consequently, it can be concluded that when employing the fine-grained time slot, approaches with equal training and inference granularities experience a decline in overall performance as granularity increases. This is primarily attributed to the unavoidable errors that are introduced by increasing the inference granularity.

Five-second-level Detection. Beyond one-second-level detection, some studies have explored stalling detection at longer time intervals, such as performing detection every five seconds or for each chunk. Within this context, we transform the time slot size to five seconds by undersampling the labels with a five-second granularity. During the evaluation phase, we evaluate performance solely on these corresponding sampled instances. Similar to the one-second-level detection task, we examine the performance trends under two scenarios as the detection granularity varies.

In scenario 1, we fix the training granularity at five seconds while varying the inference granularity from one to ten seconds. Scenario 2 maintains a constant inference granularity of five seconds and adjusts the training granularity within the same range. Figure 12 illustrates the trends of three metrics for both scenarios as the granularity changes. When the inference granularity falls below five seconds, the f1-score exhibits a decline as the inference granularity increases. A similar downward trend is observed in CR@10 and RT@10. This degradation can be attributed to the misalignment between the model's predicted samples and the evaluation samples. When the inference granularity exceeds 5 seconds, a trend is observed that is similar to the one-second-level detection, where the detection performance degrades as the granularity becomes larger. However, at a ten-second granularity, all predicted samples have corresponding evaluation instances, resulting in an improvement in performance.

In scenario 2, where the training granularity is finer than the time slot size, the training process effectively oversamples the training set at a five-second granularity. However, as depicted in Figure 12, this does not result in detection performance improvement. A one-second training granularity yields only a slight increase in f1-score (average 0.16% improvement on four traces) compared to a five-second granularity. This

TABLE VI
F1-SCORE OVER DIFFERENT WINDOW SIZES OF TWSF ACROSS THE FOUR TRACES.

Case	NTI	CLU	TLU	F1-Score	CR@10	RT@10(s)
I	×	×	×	0.7809	0.5326	6.994
II	✓	×	×	0.6437	0.2287	8.745
III	✓	✓	×	0.6153	0.2107	8.893
IV	×	✓	×	0.7447	0.4541	7.575
V	×	×	✓	0.7712	0.4566	7.362

incremental benefit is accompanied by a fourfold increase in the number of training samples. It can be concluded that oversampling in the time dimension does not significantly boost detection performance. **The optimal balance between performance and training cost is achieved when the training and inference granularities align with the time slot size.**

Chunk-level detection. Previous discussions on detection granularity have primarily focused on uniform undersampling along the time dimension. However, chunk-based detection methods can be considered as forms of non-uniform undersampling. To evaluate the impact of these two sampling approaches, we conduct an ablation study on Trace A. We employ the CSF as our baseline feature set and introduce three distinct scenarios (NTI, CLU and TLU) for processing it. **1) NTI**. In this scenario, we remove all time-interval-related features from the set, retaining only four features (*chunk size*, *download time*, *IRT* and *IDET*) that describe each chunk. This reduces the feature dimension to 240. Consequently, the chunk sequence only preserves the chunk attributes, resulting in identical features across different time slots within the same chunk. This can be interpreted as an oversampling of the sequence features by a factor of n, where n corresponds to the chunk's IDET. **2) CLU**. CLU represents the utilization of chunk-level undersampling for the training set. A non-uniform undersampling is applied to the samples within each session, with only those samples retained at the time slots when chunks download end for model training. When both NTI and CLU are employed, this process effectively removes duplicates from the training set. **3) TLU**. In Trace A, chunk-level undersampling typically preserves approximately 15% of the samples. For comparison, we introduce TLU, which employs a uniform sampling scheme with both training and inference granularities set to 7 seconds.

A series of evaluations are conducted on Trace A to assess the impact of various combinations of the three scenarios. The results, presented in Table VI, demonstrate that the removal of time interval-related features has a significant negative effect on detection performance. This results in a 13.7% decrease in f1-score, a 30.4% reduction in CR@10, and a 25% increase in response time (Case I vs. Case II). The introduction of chunk-level detection further diminishes performance, with the f1-score dropping from 64.4% to 61.5%. Furthermore, we compare the performance of chunk-level detection (Case IV) with that of fixed detection granularity (Case V) under similar sampling ratios. The results demonstrate that uniform undersampling outperforms non-uniform undersampling in terms of accuracy and sensitivity (77.1% vs. 74.5% in f1-score and 45.7% vs 45.4% in CR@10). **This suggests that detecting stalls per chunk may not be the most optimal approach compared to uniform undersampling along the time dimension.**

D. Experiment for the Sample Reweighting Method

Next, we investigate the impact of the DBR method on detection performance. The DBR method is integrated into various detection feature extraction methods, including those presented in Orsolic's work, ViCrypt, Requet, and our proposed selected feature sets. The results of the ablation study indicate that the combination of $\gamma = 0.1$ and $L = 20s$ represents the optimal configuration in terms of overall performance. Subsequently, the detection performance is evaluated under two scenarios: one utilizing the DBR method and another without it.

Figure 13 presents a comparison of four feature extraction methods across four traces. From the perspective of f1-score, the DBR method performs comparably to the original detection methods in most scenarios. However, the DBR method exhibits improvements in sensitivity metrics. Specifically, models that employ DBR achieve an average increase of 7.2% in CR@10 and an 8.0% reduction in response time in comparison to models that do not employ DBR. This enhancement can be attributed to DBR's tendency to prioritize learning from samples close to stall change events, thereby increasing detection sensitivity.

Parameter Study of DBR. We tested various parameter configurations for γ and L, as well as different ways of decreasing the parameters with respect to the switch distance, to evaluate their impact on detection performance. Specifically, we explored $\gamma \in \{0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}$ and $L \in \{1s, 3s, 5s, 10s, 20s\}$ on Trace A. In addition to the exponential function defined in Equation 8, we introduced four other functions to define the decay behavior: the *stage function*, the *linear function*, the *root function*, and the *geometric function*.

- The stage function is a two-stage discrete function where samples with small distance are distributed high weight and samples with long distance low weight.

$$\lambda_{\text{stage}}(l) = \begin{cases} 1, & l \le L \\ \gamma, & \text{others} \end{cases}. \tag{9}$$

- The linear function employs a constant drop rate of the sample weight when the distance $l \le L_0$.

$$\lambda_{\text{linear}}(l) = \max\left(\gamma, 1 - \frac{1-\gamma}{L} \cdot l\right). \tag{10}$$

- The root function utilizes a drop rate of the sample weight to be inversely proportional to the sample switch distance: $\frac{\partial \lambda(l)}{\partial l} = -\frac{P}{\partial l}$, where $P \ge 0$ is a constant.

$$\lambda_{\text{root}}(l) = \max\left(\gamma, 1 - \sqrt{\frac{(1-\gamma)^2}{L} \cdot l}\right). \tag{11}$$

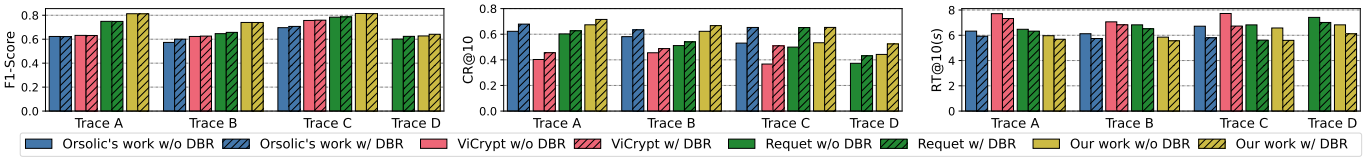


Fig. 13. The impact of adopting the DBR method on detection performance with varying feature extraction methods on four traces. As shown in the figure, regardless of the feature extraction method (Orsolic's work, ViCrypt, Requet, and our proposed feature set) used, adopting the DBR approach consistently improves both classification accuracy and detection sensitivity across all traces.

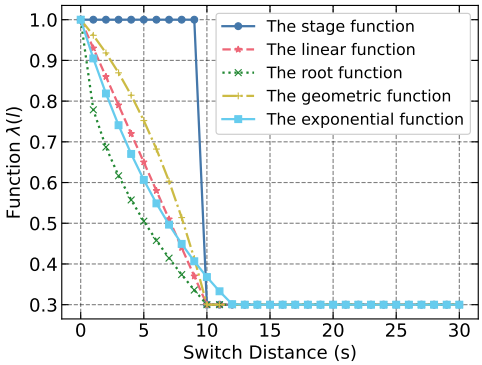


Fig. 14. Five types of reweighting functions with $L = 10s$ and $\gamma = 0.3$.

- The geometric function oppositely uses a drop rate where the smaller distance weight drops smoother.

$$\lambda_{\text{geometric}}(l) = \max\left(\gamma, 1 + \gamma - \gamma \cdot 2^{\frac{-\log_2 \gamma}{L} \cdot l}\right). \quad (12)$$

Figure 14 shows five types of function curves we design.

Figure 15 plots the performance of f1-score, CR@10 and RT@10 with different values of γ and L across five reweighting functions. We first analyze the impact of different γ values on detection performance. Regardless of the function type or the L value, we observe that the best accuracy and detection sensitivity are generally achieved when $\gamma = 0.1$. When fixing $\gamma = 0.1$, the optimal L varied depending on the function type, typically falling within the range of 3 to 10 seconds. Next, we selected the optimal γ and L combinations for each function type to compare their performance differences. Under these optimal settings, we find that the performance gap between different functions is minimal. Specifically, the difference in f1-score between the best and worst functions was only 0.5%, while the differences in CR@10 and RT@10 were 0.9% and 0.58 seconds, respectively. These results suggest that the rate of weight decay (determined by the function type) has a negligible impact on the model's detection performance. Instead, the performance differences primarily arise from the choice of γ and L values.

VII. Discussion

Looking back, the proposed detection framework and experiments have successfully achieved the objectives we set out.

Detection Granularity: By decoupling the detection granularity from the time slot size, we address the challenge of fair comparisons across different approaches. Furthermore, the framework unifies chunk-level detection and time-slot-level detection, considering their interactions across training granularity, inference granularity, and time slot size. While chunk-level detection can be viewed as a type of non-uniform sampling with slightly weaker detection performance compared to uniform sampling with a similar sampling ratio, this method emphasizes lightweight deployment. By constructing features solely for each chunk, the computational cost of feature extraction is significantly reduced.

The separation of training and inference granularity also provides valuable insights for optimizing resource utilization when deploying detection models. For example, we can increase the training granularity to reduce the amount of training data, thereby saving data collection and training resources with only a slight decrease in detection accuracy, while deploying the model at a finer inference granularity. Alternatively, we can train a detection model with a fixed training granularity and deploy it at different inference granularities, thereby enabling model reuse and reducing the need to train separate models for each deployment scenario. These strategies provide practical ways to balance detection performance and resource efficiency in real-world deployments.

Feature Construction: The ultimate goal of this work is not to propose the most optimal feature extraction solution that guarantees superior classification performance but rather to provide a novel perspective for constructing features tailored to stalling detection. We have validated that fusing packet-level and chunk-level features yields performance gains. Furthermore, incorporating sequence-based features demonstrates that performing simple statistical calculations (e.g., mean) over time windows and extending sequence lengths can significantly enhance results. Conversely, in scenarios where feature quantity must be constrained, we propose an effective method to reduce features without causing substantial performance degradation by shortening the sequence length. For example, halving the sequence length incurs minimal performance loss. However, excessive reduction in sequence length leads to a sharp decline in performance.

Notably, temporal dependencies between windows do not require explicit feature modeling; instead, classifiers can learn the sequential order of feature windows independently. This approach circumvents the need for large-window feature construction, as seen in methods like Requet, and allows focus on efficient feature extraction within smaller windows. Future research could explore using neural networks to model such sequence features. While direct applications of RNNs or Transformers to sequences in this study did not yield

This article has been accepted for publication in IEEE Transactions on Network and Service Management. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2025.3554822

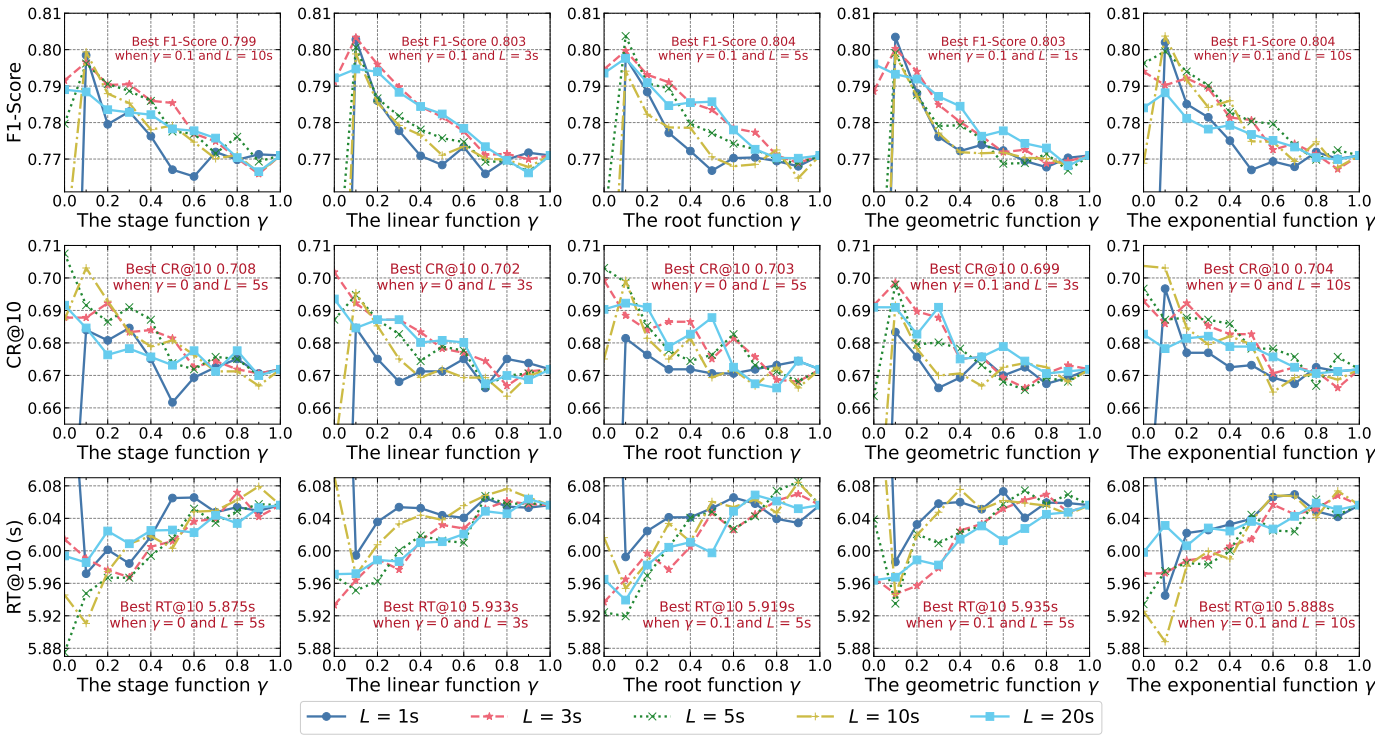


Fig. 15. Detection performance of different reweighting functions on the trace A under varying parameter combinations of L and γ.

TABLE VII
COMPARISON OF METHODS.

Aspect	Our work	ViCrypt	Requet
Time slot size	**One second**	**One second**	Five seconds
Feature types	**Chunk-level & packet-level**	Packet-level	Chunk-level
Trace types	**VoDs & live streaming**	VoDs	VoDs
F1-score of stalling states	**81%**	63%	75%
CR@10 of stalls	**67%**	40%	60%
Flexible inference granularity	**Yes**	No	No
Number of features	810 or 200	208	**120**

significant performance improvements, there remains potential in this area.

Sensitivity Metrics for Stalling Detection: We introduce sensitivity metrics to evaluate the timeliness of detecting complete stalling events. Unlike previous studies that focus on overall classification performance, this approach aligns more closely with the core objective of stalling detection. Sensitivity metrics enable a more comprehensive assessment of detection methods without being disproportionately influenced by extremely long stalling events.

These metrics increase the focus on short stalling events, which aligns with practical intuition. For long stalling events: If the model fails to detect them for an extended period, users are unlikely to wait indefinitely for playback to resume. If the model promptly detects a stalling event, network optimization measures can be applied to improve conditions, often breaking the continuity of the stalling state. Thus, our primary concern is not every time slot within a prolonged stalling event but rather the seconds before and after the stalling occurrence.

Advantages of the DBR Method The origin of the DBR method lies in the concept of estimating event states directly, offering a fresh perspective on the stalling state classification problem. Sensitivity metrics provide complementary insights into classification performance, particularly emphasizing timeliness. Notably, across different feature extraction methods, we observe a strong correlation between classification accuracy and detection sensitivity. Integrating the DBR method further enhances both metrics, underscoring its effectiveness in improving stalling detection performance.

Finally, we compare our method with existing stalling detection methods across several key aspects. As shown in Table VII, our method outperforms others in terms of the scope of traces, the size of the time slot, the comprehensiveness of the stalling metric, and the classification performance of the model. Specifically, our approach is more flexible with respect to different types of traces, uses a one-second time slot size for finer-grained detection, incorporates both chunk-level and packet-level features, and achieves superior classification accuracy. However, one limitation of our method is that it requires a larger number of features, which increases the resource

consumption during the feature extraction phase. This trade-off is necessary for achieving the high detection performance, but it may not be suitable for resource-constrained environments. Despite this, we believe that the benefits of our approach in terms of detection accuracy and timeliness make it a strong candidate for practical applications where resource availability is less of a concern.

VIII. Conclusion

We present a modelling approach to video stalling events using ML models, evaluated on three publicly available datasets as well as our collected YouTube trace. First, a sequence-based feature extraction method is proposed to achieve more accurate stalling detection due to long-term pattern extraction and feature fusion across different layers. Second, we introduce the time slot size, training granularity, and inference granularity to enable the analysis at different temporal resolutions. We find that increasing the inference granularity is the main reason for the degradation of detection classification and sensitivity performance. In contrast, increasing or decreasing the fixed training granularity does not have a large impact on performance, which is equivalent to uniformly undersampling or oversampling the training samples in the time dimension. Furthermore, compared to detecting stalling states when new video chunks arrive, using equivalent uniformly time-window-based inference yields better performance. Finally, we model the task of detecting stalls starting or ending to estimate the distribution of the prediction errors. We are able to obtain a sample error rate in the stalling detection task that is positively correlated with the temporal distance of the sample from the stalling event state. From this, we propose the distance-based reweighting method (DBR) to mine difficult samples and let the model focus on samples around stalls starting and ending. The proposed method is able to yield the sensitivity performance without classification performance degradation. This provides insights from both sample sampling and weighting perspectives to provide accurate and timely stalling detection models for lightweight inference.

References

[1] P. Jonsson *et al.*, "Ericsson mobility report," *Ericsson: Stockholm, Sweden*, 2022.

[2] L. Skorin-Kapov, M. Varela, T. Hoßfeld, and K.-T. Chen, "A survey of emerging concepts and challenges for qoe management of multimedia services," *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, vol. 14, no. 2s, pp. 1–29, 2018.

[3] A. A. Barakabitze, N. Barman, A. Ahmad, S. Zadtootaghaj, L. Sun, M. G. Martini, and L. Atzori, "Qoe management of multimedia streaming services in future networks: A tutorial and survey," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 1, pp. 526–565, 2019.

[4] V. Aggarwal, E. Halepovic, J. Pang, S. Venkataraman, and H. Yan, "Prometheus: Toward quality-of-experience estimation for mobile apps from passive network measurements," in *Proceedings of the 15th Workshop on Mobile Computing Systems and Applications*, 2014, pp. 1–6.

[5] P. Casas, M. Seufert, and R. Schatz, "Youqmon: A system for online monitoring of youtube qoe in operational 3g networks," *ACM SIGMETRICS Performance Evaluation Review*, vol. 41, no. 2, pp. 44–46, 2013.

[6] M. Seufert, P. Casas, N. Wehner, L. Gang, and K. Li, "Features that matter: Feature selection for on-line stalling prediction in encrypted video streaming," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2019, pp. 688–695.

[7] F. Loh, F. Wamser, C. Moldovan, B. Zeidler, D. Tsilimantos, S. Valentin, and T. Hoßfeld, "Is the uplink enough? estimating video stalls from encrypted network traffic," in *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2020, pp. 1–9.

[8] C. Gutterman, K. Guo, S. Arora, T. Gilliland, X. Wang, L. Wu, E. Katz-Bassett, and G. Zussman, "Requet: Real-time qoe metric detection for encrypted youtube traffic," *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, vol. 16, no. 2s, pp. 1–28, 2020.

[9] S. Wassermann, M. Seufert, P. Casas, L. Gang, and K. Li, "Vicrypt to the rescue: Real-time, machine-learning-driven video-qoe monitoring for encrypted streaming traffic," *IEEE Transactions on Network and Service Management*, vol. 17, no. 4, pp. 2007–2023, 2020.

[10] F. Bronzino, P. Schmitt, S. Ayoubi, G. Martins, R. Teixeira, and N. Feamster, "Inferring streaming video quality from encrypted traffic: Practical models and deployment experience," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 3, no. 3, pp. 1–25, 2019.

[11] S. C. Madanapalli, A. Mathai, H. H. Gharakheili, and V. Sivaraman, "Reclive: Real-time classification and qoe inference of live video streaming services," in *2021 IEEE/ACM 29th International Symposium on Quality of Service (IWQOS)*. IEEE, 2021, pp. 1–7.

[12] V. Krishnamoorthi, N. Carlsson, E. Halepovic, and E. Petajan, "Buffest: Predicting buffer conditions and real-time requirements of http (s) adaptive streaming clients," in *Proceedings of the 8th ACM on Multimedia Systems Conference*, 2017, pp. 76–87.

[13] S. C. Madanapalli, A. Mathai, H. H. Gharakheili, and V. Sivaraman, "Modeling live video streaming: Real-time classification, qoe inference, and field evaluation," *arXiv preprint arXiv:2112.02637*, 2021.

[14] F. Loh, A. Pimpinella, S. Geißler, and T. Hoßfeld, "Uplink-based live session model for stalling prediction in video streaming," in *NOMS 2023-2023 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2023, pp. 1–9.

[15] O. Belmoukadam and C. Barakat, "Unveiling the end-user viewport resolution from encrypted video traces," *IEEE Transactions on Network and Service Management*, vol. 18, no. 3, pp. 3324–3335, 2021.

[16] X. Zhang, G. Xiong, Z. Li, C. Yang, X. Lin, G. Gou, and B. Fang, "Traffic spills the beans: A robust video identification attack against youtube," *Computers & Security*, vol. 137, p. 103623, 2024.

[17] M. H. Mazhar and Z. Shafiq, "Real-time video quality of experience monitoring for https and quic," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2018, pp. 1331–1339.

[18] I. Orsolic and L. Skorin-Kapov, "A framework for in-network qoe monitoring of encrypted video streaming," *IEEE Access*, vol. 8, pp. 74 691–74 706, 2020.

[19] F. Loh, F. Poignée, F. Wamser, F. Leidinger, and T. Hoßfeld, "Uplink vs. downlink: Machine learning-based quality prediction for http adaptive video streaming," *Sensors*, vol. 21, no. 12, p. 4172, 2021.

[20] A. Ahmed, Z. Shafiq, H. Bedi, and A. Khakpour, "Suffering from buffering? detecting qoe impairments in live video streams," in *2017 IEEE 25th International Conference on Network Protocols (ICNP)*. IEEE, 2017, pp. 1–10.

[21] T. Guarnieri, I. Drago, A. B. Vieira, I. Cunha, and J. Almeida, "Characterizing qoe in large-scale live streaming," in *GLOBECOM 2017-2017 IEEE Global Communications Conference*. IEEE, 2017, pp. 1–7.

[22] M. Siekkinen, T. Kämäräinen, L. Favario, and E. Masala, "Can you see what i see? quality-of-experience measurements of mobile live video broadcasting," *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, vol. 14, no. 2s, pp. 1–23, 2018.

[23] T. Zhang, F. Ren, and B. Wang, "Modeling and analyzing live streaming performance," in *2020 IEEE/ACM 28th International Symposium on Quality of Service (IWQoS)*. IEEE, 2020, pp. 1–10.

[24] T. Zhang, Z. Tang, J. Bao, and F. Ren, "Towards impact of chunk-level characteristics on mobile live streaming performance," *IEEE Transactions on Mobile Computing*, 2022.

[25] X. Zhu, S. Sen, and Z. M. Mao, "Livelyzer: analyzing the first-mile ingest performance of live video streaming," in *Proceedings of the 12th ACM Multimedia Systems Conference*, 2021, pp. 36–50.

[26] F. Loh, K. Hildebrand, F. Wamser, S. Geißler, and T. Hoßfeld, "Machine learning based study of qoe metrics in twitch. tv live streaming," in *NOMS 2023-2023 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2023, pp. 1–7.

[27] M. Shen, J. Zhang, K. Xu, L. Zhu, J. Liu, and X. Du, "Deepqoe: Real-time measurement of video qoe from encrypted traffic with deep learning," in *2020 IEEE/ACM 28th International Symposium on Quality of Service (IWQoS)*. IEEE, 2020, pp. 1–10.

[28] F. Loh, F. Wamser, F. Poignée, S. Geißler, and T. Hoßfeld, "Youtube dataset on mobile streaming for internet traffic modeling and streaming analysis," *Scientific Data*, vol. 9, no. 1, p. 293, 2022.
[29] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 2016, pp. 785–794.
[30] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.
[31] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine learning in python," *the Journal of machine Learning research*, vol. 12, pp. 2825–2830, 2011.

Ximin Li received the B.E. degree in electronic and information engineering from the University of Science and Technology of China, Hefei, China, in 2018, where he is currently pursuing the Ph.D. degree with the School of Information Science and Technology. His work focuses on machine learning and deep learning for networks, and QoE monitoring and modeling.

Xiaodong Xu received the B.E. and Ph.D. degrees in electronic and information engineering from the University of Science and Technology of China (USTC) in 2000 and 2007, respectively. From 2000 to 2001, he served as a Research Assistant with the R&D Center, Konka Telecommunications Technology. Since 2007, he has been a Faculty Member with the Department of Electronic Engineering and Information Science, USTC, where he is currently working with the CAS Key Laboratory of Wireless-Optical Communications. His research interests include the areas of wireless communications, signal processing, wireless artificial intelligence, and information-theoretic security.

Guo Wei received the B.S. degree in electronic engineering from the University of Science and Technology of China (USTC), Hefei, China, in 1983, and the M.S. and Ph.D. degrees in electronic engineering from the Chinese Academy of Sciences, Beijing, China, in 1986 and 1991, respectively. He is currently a Professor with the School of Information Science and Technology, USTC. His current research interests include wireless and mobile communications, wireless multimedia communications, and wireless information networks.

Xiaowei Qin received the B.S. and Ph.D. degrees from the Department of Electrical Engineering and Information Science, University of Science and Technology of China (USTC), Hefei, China, in 2000 and 2008, respectively. He has been a Staff Member with the Key Laboratory of Wireless-Optical Communications, Chinese Academy of Sciences, USTC, since 2014. His research interests include optimization theory, service modeling in future heterogeneous networks, and wireless artificial intelligence in mobile communication networks.