ViChaser: Chase Your Viewpoint for Live Video Streaming with Block-Oriented Super-Resolution

Ning Chen, Sheng Zhang, *Member, IEEE,* Zhi Ma, Yu Chen, Yibo Jin, Jie Wu, *Fellow, IEEE,* Zhuzhong Qian, *Member, IEEE,* Yu Liang, and Sanglu Lu, *Member, IEEE,*

Abstract—The usage of live streaming services has led to a substantial increase in live video traffic. However, the perceived quality of experience of users is frequently limited by variations in the upstream bandwidth of streamers. To address this issue, several adaptive bitrate (ABR) algorithms have been developed to mitigate bandwidth variations. Nevertheless, the ability of users to enjoy high-quality live streams remains limited. While neural-enhanced approaches, such as super-resolution, offer significant quality improvements, frame-oriented super-resolution leads to excessive inference delay that violates the real-time feature of live streaming. In response, we propose ViChaser, which examines block-oriented super-resolution for live streaming. ViChaser performs neural super-resolution on potential blocks of interest in the media server, corresponding to the user's viewpoint, and uses online learning to adapt to the dynamic content of the video. Additionally, ViChaser utilizes the Lyapunov framework to efficiently allocate uplink bandwidth for original low-quality live video and high-quality labels. The experimental results demonstrate that ViChaser achieves 1.2-1.5 dB higher video quality in Peak-Signal-to-Noise-Ratio than WebRTC and increases processing speed by 11-16 fps relative to LiveNAS.

Index Terms—block-oriented super-resolution, live video streaming, online learning.

1 Introduction

The proliferation of live streaming platforms, such as YouTube Live [1] and Twitch.tv [2], has led to a significant surge in the volume of live video traffic, resulting in a considerable share of the total Internet video traffic. As per the growth trend, Cisco has projected that live video traffic may occupy 27% of the total Internet video traffic in 2023 [3]. This growth in live video traffic has led to an increase in user demand for high-quality video, thereby making it imperative to support high-definition (i.e., 1080p) streaming on mobile devices to stay competitive.

Typically, a complete live streaming system comprises a streamer, a media server, and numerous Content Delivery Network (CDN) nodes, as depicted in Fig.1. The streamer is responsible for capturing the live video and delivering it in real-time from the source streamer to the centralized media server, using low-latency streaming protocols such as RTSP [4] and RTP [5]. Subsequently, the media server transcodes the received high-quality raw videos, such as 1080p or 4k, into several lower-quality versions, such as 360p and 270p, for each live stream. Finally, the CDN nodes distribute the requested videos to millions of concurrent mobile users. It is crucial to note that upscaling low-quality videos, such as 360p, to high-quality videos, such as 1080p, via high-quality encoding is not recommended since it is time-consuming. Generally, users demand personalized video quality, and therefore, live videos with various quality versions must be readily available at all times.

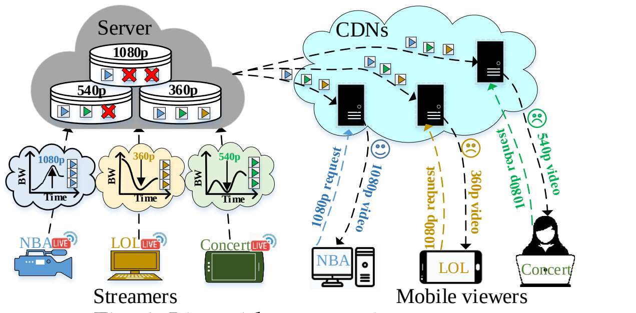


Fig. 1: Live video streaming system.

However, due to the time-varying streamer's uplink bandwidth, current live streaming system can hardly guarantee such availability, especially for the ultra high-quality videos (e.g., 1080p and 4k).

In situations where the uplink network is congested, streamer clients may have to deliver low-quality videos to the media server to satisfy real-time requirements. As a consequence, the media server can only transcode several lower-quality videos, leading to the exclusion of high-quality videos and limiting the overall quality level of the delivery. As a result, despite having abundant downlink bandwidth, mobile users are unable to enjoy high-quality live videos. For instance, as shown in Fig. 1, the user's request for 1080p NBA (National Basketball Association) videos is satisfied, but when the user requests 1080p LOL (League of Legends) videos, the media server returns only 360p videos, largely due to the NBA streamer's current sufficient uplink bandwidth and the LOL streamer's insufficient bandwidth. It is crucial to note that attempting high-quality encoding in such a dilemma may cause unacceptable transmission delays, leading to a degradation in the user's

- *N. Chen, S. Zhang, Z. Ma, Y. Chen, Y.B. Jin, Z.Z. Qian, and S.L. Lu are with the State Key Laboratory for Novel Software Technology, Nanjing University, China. E-mail: {sheng, qzz, sanglu}@nju.edu.cn,{ningc,yibo.jin,dz1733013,dz1933005}@smail.nju.edu.cn.*
- *J. Wu is with the Center for Networked Computing, Temple University, Philadelphia, PA 19122, USA. E-mail: jiewu@temple.edu.*
- *Yu Liang is with the School of Computer and Electronic Information/School of Artificial Intelligence, Nanjing Normal University, China. E-mail: liangyu@njnu.edu.cn.*
- *The corresponding author is Sheng Zhang (sheng@nju.edu.cn). This work was supported in part by NSFC (61872175, 61832008), and Collaborative Innovation Center of Novel Software Technology and Industrialization.*

quality of experience. Therefore, the primary focus of this paper is to address how to alleviate the impact of uplink bandwidth variation on the available quality levels in the media server.

To address the issue of uplink bandwidth variation, the proposed solution involves the use of adaptive bitrate (ABR) streaming algorithms [6] to enable the streamer client to select the most suitable bitrate that matches the available bandwidth. Subsequently, the media server leverages mathematical interpolation techniques such as bilinear in WebRTC [35] to generate high-quality videos. However, this method has limitations in utilizing the specific characteristics of live video, resulting in obscure quality gains.

The use of deep neural network (DNN) inference on video frames has recently been proposed as a promising technique to improve live video quality, known as neural-enhanced video delivery [37]. However, previous methods have performed DNN reconstruction on the entire frame, leading to excessive inference delay that affects the real-time nature of live video. In this study, we propose an alternative approach where DNN super resolution is performed on smaller input sizes, which significantly reduces inference time. Furthermore, we observe that mobile users tend to be interested in specific blocks within each frame that contain relevant events or objects, such as basketball players and moving balls in NBA games. Hence, we suggest aligning the perceived quality of each frame with these dynamic user interests, while maintaining lower quality in other areas of the frame. Our experimental results, presented in Sec. 2.2, support the effectiveness of this approach.

In this paper, we introduce a novel approach, named ViChaser, for enhancing the resolution of live video frames using content-aware block-enhanced neural super-resolution. Instead of continuously transmitting high-quality videos, ViChaser utilizes neural super-resolution to generate multiple quality versions from low-quality videos delivered to the media server. The proposed method employs DNN super-resolution only on potential blocks of interest in live video frames, rather than on the entire frame. The pre-trained approach for super-resolution is found to be unsuitable for live video streaming with frequent scene changes, as demonstrated in Sec. 2.3. Therefore, ViChaser adopts an online learning approach to continuously train its DNNs with fresh data. The simultaneous execution of the online learning module and block-enhanced neural super-resolution is a non-trivial task. Our results indicate that ViChaser outperforms existing methods in terms of image quality and processing time.

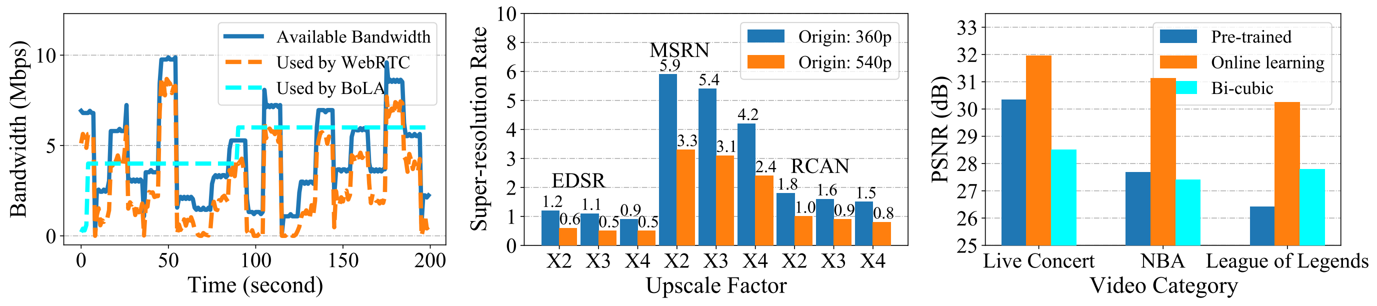
Implementing ViChaser is never a trivial task, as it must overcome three critical challenges. Firstly, accurately and timely predicting potential blocks of interest can be difficult. While learning-based methods such as [19] can achieve high accuracy, they often incur significant delays. Other methods, such as [39], often fail to precisely identify target blocks. Secondly, a large number of blocks may be generated from a single frame, but the constrained bandwidth does not allow for their real-time transmission to the media server. Thirdly, maximizing the user-perceived Quality of Experience (QoE) requires efficient allocation of the available bandwidth for video and label transmission based on the runtime performance of online learning.

ViChaser proposes to address three challenges in live video streaming through the development of innovative functional modules at both the streamer client and media server. The first challenge is tackled by utilizing the tensorRT yolov5 algorithm in the media server to identify potential blocks by generating bounding boxes from the original low-quality video. The second challenge is addressed by generating high-quality labels in the streamer client using raw frames and detecting results returned from the media server. To optimize this process, ViChaser filters redundant raw frames using the EdgeDiff filter and prioritizes labels based on their sizes. The final challenge of limited uplink bandwidth is managed by transmitting low-quality live videos and high-quality labels to maximize user-perceived video quality without compromising the real-time nature of the stream. The Lyapunov framework is employed to determine the quality version of the transmitted live video and the quantity of high-quality labels, without relying on future information, including uplink network bandwidth and video content.

As acknowledged by ViChaser, reducing the input size can have a substantial impact on accelerating deep neural network super-resolution. This approach is not novel, as demonstrated by LiveNAS [38] and Supremo [39], which adopt different strategies to parallelize super-resolution using multiple GPUs and offloading selected blocks to a media server for processing, respectively. However, the methods employed by these two techniques for sketching blocks introduce various challenges. For instance, LiveNAS implemented a uniform division of videos into multiple slices. This approach may result in a dispersion of target objects across several adjacent slices, ultimately leading to variations in quality among these slices. Moreover, its parallel approach necessitates considerable GPU resources that may not always be available. On the other hand, Supremo relies solely on the Canny edge detector in OpenCV [46] for selecting blocks, which can lead to misidentifying background regions as target blocks, thus increasing computational complexity.

We evaluate ViChaser using a prototype implementation. We collect three categories of recorded live video streams (e.g., live Concert, live NBA game and live LOL game). For each category, we sample 5 videos with a total duration of about 100 minutes, and each video has three resolution versions (i.e., 270p, 360p, and 540p). The experimental results show that ViChaser improves video quality by 1.2-1.5 dB over WebRTC in PSNR, and increases the processing speed by 11-16 fps compared with LiveNAS. We summarize our major contributions of this work as follows:

- Timely and accurate target blocks prediction. We propose lightweight *EdgeDiff Filter* to filter out massive redundant frames, and use learning-based detector to predict the final blocks, which significantly improves the efficiency of block generation.
- Priority queue for label transmission. We set priorities for these candidate labels by their pixel sizes, which potentially improves the video quality by utilizing more labels for online learning.
- Efficient bandwidth allocation. The proposed *Transmission Controller*, which schedules the available bandwidth for low-quality video and high-quality label transmission,



(a) Uplink bandwidth variation and usage (b) DNN inference rate on different input size (c) Content-adaptive DNN super-resolution

Fig. 2: Motivations of ViChaser. (a) Bandwidth usages of WebRTC and BoLA over the time; (b) Inference rates (i.e., fps) of EDSR, MSRN and RCAN with different input resolutions and upscale factors; (c) Achieved PSNR (i.e., Peak Signal to Noise Ratio) when using pre-trained model, online model and bicubic interpolation to upscale three video categories.

leverages the *Lyapunov* framework to maximize the overall video quality while ensuring the real-time streaming.

2 Motivation

We start by showing that existing approaches to solve the streamer's uplink bandwidth constraint can hardly satisfy users' diverse demands. We then present our preliminary works to prove that frame-enhanced neural super resolution is time-consuming and hurts the real-time feature of live video streaming. Finally, we show that the proposed block-enhanced super resolution method needs to adapt to the video content via online learning.

2.1 Prior Approaches to Resolve Uplink Variation Meet Bottlenecks

The time-varying uplink bandwidth from the streamer to media client has a significant impact on the quality of a live video stream and its real-time feature. As stated before, the media server is only capable of transcoding the raw video (i.e., 540p) into the lower-quality (i.e., 270p) videos, hence the media server lacks the high-quality (i.e., 1080p or 4k) videos, thereby degrading the quality of experience (QoE) of downstream users. However at each transmission period, the streamer client cannot make aggressive use of the available bandwidth to transmit high-quality video because 1) it needs to reserve partial bandwidth to retransmit the losing packets and 2) it has to meet the real-time property of live streams, hence to avoid delay backlog it cannot allocate the bitrate that exceeds the available bandwidth. We summarize two kinds of prior solutions:

ABR-based schemes: Adaptive bitrate (ABR) algorithms become the primary tool that the streamer client uses to enhance video quality. ABR algorithms dynamically select a bitrate for each chunk based on the estimated network throughput and buffer occupancy. Mostly, ABR algorithm is used in the downstream, but it also works in the upstream. We collected an ATT-LTE network bandwidth trace from the Mahimahi [7] project and depicted them as Fig. 2a shows. Then, we ran two ABR algrithms (e.g., rate-based WebRTC [35] and buffer-based BoLA [31]) under this bandwidth, and obtained the bitrate selections. Compared to BoLA, WebRTC has a much more conservative mechanism to use the bandwidth. However, WebRTC is hindered by the inevitable biases present when estimating the network throughput on top of HTTP. Even thought several smoothing heuristics [8] are used to assist the throughput estimation, accurate throughput prediction remains a nonnegligible challenge. BoLA adopts a large buffer, which can effectively absorb bandwidth variation, but the display in downstream client is lagging behind.

The above ABR-based schemes mitigate the uplink bandwidth variation and ensure the real-time video streaming, nonetheless, using these schemes alone cannot meet the needs of users for high quality video when facing terrible networks. When the available bandwidth is insufficient, the ABR algorithms are likely to select a low bitrate, which delegates the highest quality version of videos stored at the media server, thereby restricting the quality of the entire downstream. For instance, WebRTC assigns 2 Mbps for next video chunk, which may generate up to 540p video, hence the users who request for 1080p video can only receive at most 540p video. In such case, several interpolation-based upsampling methods [9], [10], [11] such as nearest-neighbor interpolation, bicubic interpolation, and bilinear interpolation are proposed to upscale the videos. However, the interpolation-based approaches improve the image resolution based largely on its own image signals, without considering the video characteristics, hence it achieves obscure quality gains. Instead, it often brings some side effects, such as noise amplification, computational complexity, and blurring results. Hence, pure interpolation-based method can hardly meet the quality requirement.

Neural super-resolution schemes: Recent studies [41], [42], [43] prove that deep neural networks (i.e., DNNs) are helpful in learning the mapping from a low resolution to high resolution, and many further advancement [37], [38], [44], [45] to improve the inference and training. NAS [37] adopted DNNs super-resolution in the downstream. It trains a super-resolution DNN for each video offline in media server, and sends the corresponding DNN based the requested video. The client performs super-resolution in the received low-quality video by utilizing the local computation. Indeed, NAS can hardly improve the quality of live video because it uses a pre-trained DNN model. Hence, LiveNAS [38] proposes an online method to learn the adaptive DNNs and maximizes the quality gain. However, both NAS and LiveNAS perform DNNs super-resolution on the whole frame, which causes unpredictable delay. We demonstrate this with preliminary testing results in

TABLE 1: Time overhead of performing super-resolution

SR Model	270p (s)	360p (s)	540p (s)	720p (s)	PSNR (dB)	SSIM (none)
EDSR X2	0.379	0.853	1.535	4.710	33.92	0.9013
EDSR X3	0.428	0.961	1.747	5.034	29.25	0.8093
EDSR X4	0.482	1.096	1.965	5.537	27.71	0.7420
EUSR X2	0.069	0.152	0.267	0.921	31.92	0.9105
EUSR X3	0.075	0.165	0.281	0.964	28.57	0.8376
EUSR X4	0.086	0.218	0.354	1.099	26.21	0.7834
CARN X2	0.023	0.050	0.089	0.282	31.92	0.9256
CARN X3	0.033	0.072	0.125	0.358	28.06	0.8493
CARN X3	0.047	0.098	0.171	0.469	26.07	0.7837
MSRN x2	0.075	0.167	0.302	0.922	32.22	0.9245
MSRN x3	0.081	0.182	0.321	1.003	29.08	0.8581
MSRN x4	0.091	0.237	0.404	1.118	26.39	0.7921
RCAN x2	0.269	0.595	1.057	3.191	33.34	0.9384
RCAN x3	0.277	0.610	1.085	3.261	29.09	0.8702
RCAN x4	0.287	0.632	1.149	3.356	26.82	0.8087

Sec. 2.2. When generating 4k live video, LiveNAS splits up the frame into several slices and performs slice-parallel super-resolution using multiple servers. Nonetheless, the strict computation requirement can hardly be satisfied in the media server side, not alone in the client side. Hence, existing learning-based super resolution approaches are still not appropriate in live video.

2.2 Expensive Frame-Level Reconstruction

Undeniably, given a received low-quality video in the media server, deep learning-based super-resolution can generate high-quality video to meet user's personalized quality demand, the frame-enhanced super-resolution still faces lots of challenges in practice. By utilizing one NVIDIA RTX 2080 Ti GPU, we implemented five well-studied SR models including EDSR [12], CARN [13], MSRN [14], EUSR [15] and RCAN [16] with different upscale factors and train them using dataset BSD100 [18]. Note that we use *FFMPEG* tool to generate the labels for model training. In addition, we collect three live videos with a total of about 3 hours duration to evaluate theirs performance in terms of inference time and achieved quality. We depict the testing results as Tab. 1 shows. For the same DNNs, enlarging the input frame size incurs a significant increase in the reconstruction cost. For instance, using state-of-the-art EDSR to upscale 720p to 2880p needs an amazing time about 5.537 seconds. Even given the 270p image, it still needs 0.379 seconds to upscale it to 540p. The other models such as CARN, EUSR and MSRN show an acceptable performance when given a low resolution, but still perform poorly when enlarging the input resolution. To go a step further, based on the inference time, we calculate the inference rate in Fig. 2b. Taking the MSRN model as an example to illustrate this hidden problem, only an inference rate of 15 fps is obtained when upscaling 270p to 540p, which can hardly meet the basic fps requirement of 16 fps, let alone with a bigger upscale factor. The state-of-the-art model RCAN earns a perfect PSNR but gains a poor inference rate. In conclusion, the pure frame-enhanced super-resolution does enhance the video quality, but it incurs excessive reconstruction time, which can hardly meet the real-time feature of the live video streaming.

Can we lower the input resolution to accelerate the DNNs inference, but still obtain a perceived high-quality? A naive approach is to uniformly divide the frame into several slices and perform SR for each slice. However, such method misses the correlations between slices and is still time-consuming. Another nature but promising idea arises from users' habit when watching live videos. Rather than fix their viewpoints during the video display, users may adjust their viewpoints to chase the objects that attracting them. For example in a live NBA game, users may pay more attention to their favorite stars and the moving ball but ignore the background. In a live concert, audiences care more about the singers. In practice, it is challenging to accurately predict the viewpoints for each user, but we are able to capture the potential blocks that probably contain user's favorite objects. Hence, from the user's perspective, enhancing the video quality within these blocks brings nearly the same QoE with the frame-enhanced super-resolution. We details how to capture user's blocks of interest and perform block-enhanced super-resolution in Sections 4 and 5.

2.3 Content-Adaptive Model Updates

Generally speaking, performing DNN-based super resolution using a neural network on the similar video with training data gains great benefit due to the correlation of model and video content. However, employing a pre-trained or non-updated model may hit a bottleneck of quality gains in live streaming. Scene changes are common in many streaming session. Hence, a more feasible approach is to train the super-resolution DNNs in an online manner, namely training DNNs with previous videos from current streaming session. To verify its efficiency in quality gains than pre-training manner, we test the resulting video quality in PSNR of online learning and pre-trained means using diverse types of videos, including Concert, NBA, and LOL videos. We use BSD100 dataset [18] as training data to train MSRN model [14] and make no updates when performing super-resolution on the above videos. For online learning, we use the current streams as training data to continuously train the model, and apply the latest DNN model for super-resolution. In addition, we calculate the average quality using bicubic interpolation super-resolution. As Fig. 2c shows, compared to the pre-trained model with independent training data, the online training with fresh data achieves significant quality gains about 2 dB.

It's obvious that in our proposed scenario the fresh training data comes from the streamer client. Hence, in addition to the low-quality live videos, the streamer client is responsible for the label preparation and transmission. We detail this procedure in Section 4.

2.4 Summary of Insights

We summarize our insights as follows:

- Existing ABR-based methods do mitigate the upstream bandwidth variation, yet they are not capable of generating high-quality videos when the bandwidth is scare. The interpolation-based methods don't consider the video characteristic, thereby yielding obscure quality gains.
- Running frame-level neural super-resolution incurs expensive cost, while lowering the input resolution significantly accelerates the neural super-resolution. However, the uniform video dividing method leads to a dispersion

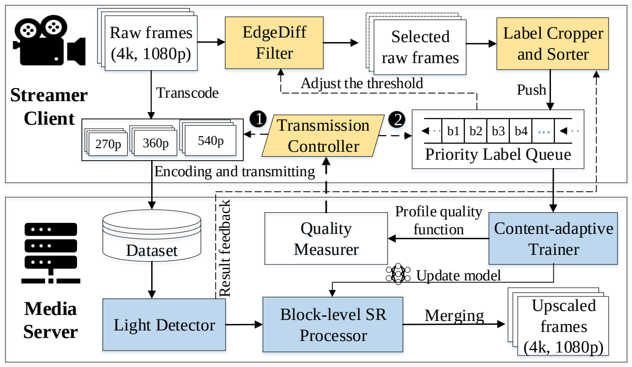


Fig. 3: System overview of ViChaser.

of target objects across several adjacent slices, causing quality variations among these slices. On the contrary, user-driven block-enhanced method accelerate the neural super resolution while maintaining a high user's perceived quality.

- Pre-trained model achieves poor quality when handling live videos with frequent scene-changes, while using online model trained with fresh data persistently obtains satisfactory quality.

3 VICHASER OVERVIEW

ViChaser aims to improve the video quality of the raw stream at the media server by running block-level super-resolution, while not degrading reliable and real-time video transmission. As Fig. 3 shows, ViChaser consists of a mobile streamer client and a GPU-provisioned media server.

The streamer client is usually deployed in the mobile device such as smart phone or camera. It first captures the raw videos with high resolution (e.g., 1080p or 4k). Rather than directly delivering them to the media server, the streamer transmits both the origin videos along with the high-quality labels, where the quality level (❶) of videos and the number of (❷) labels are determined by *Transmission Controller* based on the bandwidth estimation and the quality function of online model. As for the label generation, the streamer runs following serial operations: (1) using *EdgeDiff Filter* to filter out the redundant frames, (2) using *Label Cropper* to produce the blocks based on the feedback results from the media server, and (3) using *Sorter* to prioritize these blocks and push them into a priority queue.

The media server consists of three primary components. The *Light Detector* performs detecting algorithms (e.g., yolov5) in the arrival live videos and outputs the bounding boxes, which identify the target potential blocks of interest. Then, it sends the detecting results to the streamer client to generate high-quality labels. Meanwhile, the *Block-level SR Processor* performs neural reconstructions on the identified blocks of the current frame, and runs bicubic interpolation to upscale the low resolution to the target resolution on the rest of the frame. Finally, it merges these two parts for each frame and produces the block-enhance live video. The *Content-adaptive Trainer* performs continuously online learning using the arrival labels from the streamer client and periodical updates the DNNs for *Block-level SR processor*.

4 VICHASER STREAMER CLIENT DESIGN

As descried above, the streamer client transcodes the raw video into several low-quality versions, establishes the priority queue to generate sufficient labels for online learning. The former procedure is finished by the embedded transcoder. In this section, we first detail the labels preparation and selection, and then introduce the transmission controller to efficiently allocate the available bandwidth.

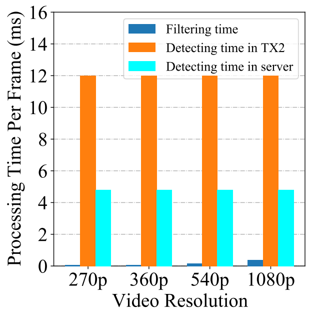
4.1 EdgeDiff Filter

Considering the content redundancy and similarity in the same streaming session, there is no need to do the same operations for each frame. Hence, we propose EdgeDiff filter, which takes the frame feature (e.g., *edge* in this paper) as the filtering metric and filters out the frames whose *edges* are below the given threshold. Then, we test its filtering time on Jetson TX2 as Fig. 4a shows, and we find that it causes only 1.9, 2.4, and 5.5 millisecond when given 270p, 360p and 540p videos respectively, which hardly hurts the real-time feature of live streaming. Note that rather than fix the *edge* threshold, we periodically adjust this threshold to keep a stable filtering rate, which ensures that the majority of representative frames of each slot are sampled. In specific, for a video with frequent scene-changes, setting a low threshold may filter out no frames and all frames are sampled, which leads to queue overflow. Therefore, EdgeDiff reduces (resp. increases) the *edge* threshold if a large (resp. small) amount of frames remain.

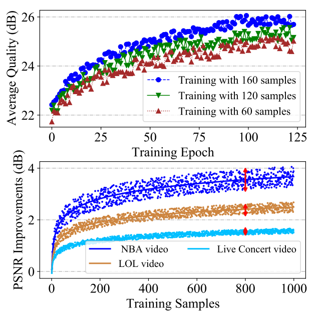
4.2 Label Cropper and Sorter

ViChaser adopts a neural learning method to upscale the interested blocks, hence the training data for online learning is also the blocks rather than the whole frame. However, the media server lacks the high-quality labels for persist training, hence ViChaser prepares labels in the streamer client. As stated before, identifying the blocks is compute-intensive using yolov5 algorithm, therefore we deploy this operation in the media server, and propose a feedback mechanism to received the detecting results. Specifically, the *Light Detector* in the media server executes yolov5 algorithm on the transmitted high-quality live video, and send back the results (e.g., bounding boxes) to the streamer client. Based on the results, the streamer client crops the corresponding blocks as the candidate labels. Note that the streamer client needs to adjust the bounding boxes to generate the labels. For example, if the resolution in the media server is 540p but 1080p in the streamer client, then the bounding boxes is to be doubled. In addition, to mitigate the detecting accuracy gap resulted from the input resolution, we slightly expand the bounding boxes to cover the entire target objects.

It is impractical to transmit all the candidate labels to the media server, especially when the live video contains frequent scene-changes. We notice that the candidate labels vary in pixel size and byte size, and block with larger pixel contains more bytes. Hence given the transmission bandwidth, more labels with smaller pixel size or less labels with larger pixel can be transmitted. To validate which one is more effective in online learning, we first divide the BSD100 dataset into two parts, one is the training set that contains



(a) Processing time



(b) PSNR improvements

Fig. 4: Processing time and quality improvements.

160 figures, and another one is the testing set that consists of 40 figures. Then, we build two comparing datasets as follows: we select 60 and 120 figures from the training set and resize them to (540,960) and (540,480) respectively. Assuming the streamer client encodes the frames with the same compression ratio, hence these two datasets need the same bandwidth (60x540x960 is equal to 120x540x480).

The testing results in the upper half of Fig. 4b show that feeding more labels brings a higher quality gain and reduces the training time. Therefore, we set priorities for these candidate labels by their pixel sizes, and the smaller (resp. larger) the block size, the higher (resp. lower) the priority. ViChaser maintain a priority queue to store theses labels. ViChaser first sorts these labels by their pixel sizes and pushes them into the priority queue and reorder this queue, and each pop operation will upload the label with the highest priority in this queue. Considering the limited queue space, ViChaser discards the remaining labels once the priority queue is full. In addition, ViChaser periodically empty the queue to store more fresh labels.

4.3 Transmission Controller

4.3.1 Optimization Objective

The goal of this module is to optimize the bandwidth allocation, which is used for transmitting the low-quality live videos and high-quality labels. Rather than directly decide the distribution shares, ViChaser controls the transmitted quality version of the live video for block-enhanced super-resolution and the quantity of high-quality labels for online learning. If a high-quality version (e.g., 1080p) is selected, there may be little bandwidth left for transmitting labels. Otherwise, more labels are utilized to enhance the online learning. To upscale to the target resolution, a larger input resolution brings a higher quality gains. For example, 1080p videos reconstructed from 540p videos show high perceived quality than that reconstructed from 360p videos. In addition, more labels benefit the online learning, thereby improving the quality gain. ViChaser aims to tradeoff them and maximizes the total quality gains. We use PSNR to represent quality for its cheaper computational overhead compared to the structural similarity index (i.e., SSIM).

Firstly, to measure the quality gap resulted from different resolution, we use $Q_{tr}^{ca}(v)$ to denote the quality gain of the live video when applying resolution v to transmit stream category ca compared to using the minimum resolution v_{min}. Specifically, we upscale the raw resolution of live video to the target resolution v_{target} through interpolation-based super-resolution, hence $Q_{tr}^{ca}(v)$ is the quality gain by upscaling v to v_{target} over upscaling v_{min} to v_{target}. We initialize $Q_{tr}^{ca}(v)$ for each version v with the achieved PSNR values using three categories of offline videos (i.e., NBA game, LOL game, and Concert). Note that $Q_{tr}^{ca}(v)$ will be persistently updated with the streams go on and this operation is finished by CPU only. On the other hand, to figure out the correlation between the number of labels and the quality improvement, we take the MSRN model trained with dataset BSD100 about 500 episodes as the initial model, and then retrain it with gradually increasing labels of above three categories of videos, and record the quality improvements and the number of training labels as the lower half of Fig. 4b shows. It is clear that more labels benefit the online learning, but the quality gains become diminished with the increase of labels. We profile this information as a function and denote it as $Q_{sr}^{ca}(n)$, where n is the label quantity and ca is the stream category. Note that $Q_{sr}^{ca}(n)$ is continuously profiled through online learning module and sent back the streamer client, which will be discussed in Sec. 5.

Two primary constraints including bandwidth and delay compress the decision space. The estimation of available uplink bandwidth, which is obtained from the underlying transport layer, is mainly used to transmit original low-quality and high-quality labels. It is apparent that if more bandwidth is used to transmit live video, less available bandwidth is left for transmitting labels, which causes little quality gains. Better balancing them is the key issue.

Now we analyze the delay overhead of ViChaser. In the streamer client side, it contains two parts including the label preparation delay l_b and transmission delay l_{tr}, where l_b is mostly incurred due to frame filtering and block cropping, and l_{tr} is caused by transmitting videos. In the media server side, it comprises the block detecting delay l_d, block-enhanced super-resolution delay l_{sr}, and merging delay l_{mer}. The above four elements make up the total processing delay L_t. To achieve a live video streaming, L_t is supposed to meet user's requirements. ViChaser aims at achieving long-term maximized video streaming quality. We divide the total time $\mathcal{T}$ into multiple slots with equal duration. Suppose that at slot time t, the selected version is v_t, and the number of labels is n_t. For simplify of illustration, we define the utility function U_t as the achieved quality of video with version v_t plus the quality gains if selecting n_t labels,

$$U_t = \omega_1 Q_{tr}^{ca}(v_t) + \omega_2 Q_{sr}^{ca}(\sum\nolimits_{i=0}^{t} n_t), \tag{1}$$

where the weighted parameters ω_1 and ω_2 balance the quality gain from the live video with version v_t and online learning with n_t labels, and $\sum \omega_i$ equals 1. $Q_{tr}^{ca}(\cdot)$ and $Q_{sr}^{ca}(\cdot)$ are two functions to indicate the quality gains derived from the live video and online learning respectively. Our objective is to maximize the time-average utility, which can be formulated as:

$$\begin{aligned}
&\mathcal{P}_1: \max_{\{v_t,n_t\}} \lim_{T\to+\infty} \frac{1}{T}\sum_{t=0}^{T}\left[\omega_1 Q_{tr}^{ca}(v_t) + \omega_2 Q_{sr}^{ca}(\sum_{i=0}^{t} n_t)\right] \\
&\text{s.t. } C_1: \sum\nolimits_{i=0}^{fps} b_{v_t,i}^{tr} + \sum\nolimits_{j=1}^{n_t} b_{n_t,j}^{label} \le B_t. \\
&\qquad C_2: \lim_{T\to+\infty} \frac{1}{T}\sum\nolimits_{t=0}^{T} L_t < L_{\max}, t \in \mathcal{T}.
\end{aligned} \tag{2}$$

Constraint C_1 ensures that the sum of bandwidth used in each slot for transmitting live videos and high-quality labels does not exceed the maximum available bandwidth B_t, in which $b^{tr}_{v_t,i}$ and $b^{label}_{n_t,j}$ represent the bytes that frame i and label j contain at slot t, respectively. Constraint C_2 guarantees the real-time video transmission, and L_{max} is the maximum acceptable delay. We set L_{max} to the length of each time slot t, such that no delay backlog is incurred.

The primary challenge that hinders the derivation of the optimal solution of problem $\mathcal{P}_1$ is the lack of future information including the accurate available bandwidth and dynamics of video content. What's more, $\mathcal{P}_1$ is an integer nonlinear programing problem and lacks efficient methods to solve it even if given priori future information. Hence, we adopt an online approach to make efficient decisions on the transmission resolution of live video and the quantity of training labels without foreseeing the future.

4.3.2 Approach

We decouple the long-term delay constraint by transforming the time-average problem $\mathcal{P}_1$ into a series of minimization problems using Lyapunov framework. Then, we develop a lightweight heuristic algorithm which only requires current bandwidth estimation without the global information over the long run. We first define a virtual queue D that records the historical measurement of the exceeded delay and set the initial queue backlog $D(0)$ to 0. The queue length evolves as

$$D(t+1) = [D(t) + L_t - L_{max}]^+ , \tag{3}$$

where $[x]^+$ is equivalent to $max\{x, 0\}$, L_{max} is the maximum acceptable delay. It is obvious that if we purse a high video quality gain by aggressively choosing a high version (e.g., 1080p) for video transmission, the queue backlog $D(t)$ may increase rapidly, which causes unacceptable transmission delay, so it is vital to keep the delay queue stable. In fact, the stability of queue D guarantees that the time-average delay does not exceed the given threshold L_{max}. In this paper ViChaser makes the decision for each slot, hence we set the L_{max} to the time length of each time slot. We then define the quadratic Lyapunov function: $L(D(t)) = \frac{1}{2}(D(t))^2$, which reflects the degree of delay queue congestion, and a small $L(D(t))$ indicates a small queue backlog. Hence, maintaining the queue stability is equivalent to push the quadratic Lyapunov function to a less congested state. To achieve it, we introduce the Lyapunov drift at slot t:

$$\Delta(D(t)) = E[L(D(t+1)) - L(D(t)) \mid D(t)]. \tag{4}$$

The drift $\Delta(D(t))$ represents the expected variation of Lyapunov function over one slot, and the smaller the $\Delta(D(t))$ is, the more stable the delay queue D is. We aim to find the optimal solution on version selection and label quantity determination. By incorporating delay queue stability into the weighted quality gain by live video with the target version and online learning with the target number of labels, we define the Lyapunov drift-plus-penalty (i.e., DPP) term as:

$$\Delta(D(t)) - V \cdot E\left[\omega_1 Q^{ca}_{tr}(v_t) + \omega_2 Q^{ca}_{sr}(\textstyle\sum_{i=0}^{t} n_t) \mid D(t)\right], \tag{5}$$

where V is a positive constant used to control the tradeoff between the delay minimization and utility maximization. Instead of minimizing the DPP term directly for each slot, Lyapunov optimization framework aims at minimizing its upper bound. In our proposed scenario, we derived an upper bound in the following lemma:

Algorithm 1: Min-DPP Algorithm

```
Input: $D(0) \leftarrow 0, \omega_1, \omega_2, L_{max}, fps, \mathcal{V}$
Output: $v_t$ and $n_t$ $\forall t \in \mathcal{T} = \{1, \dots, T\}$
1  for $t = 1$ to $T$ do
2      Initialize version $v'$ to the highest version in $\mathcal{V}$;
3      Initialize DPP value $dpp'$ to $+\infty$;
4      Profile quality gain function $Q^{ca}_{sr}(\cdot)$ and $Q^{ca}_{tr}(\cdot)$;
5      Obtain the bandwidth estimation $B_t$;
6      repeat
7          $n' \leftarrow \arg\max_{n_t} \left\{ \sum_{i=0}^{fps} b^{tr}_{v_t,i} + \sum_{j=1}^{n_t} b^{label}_{n_t,j} \le B_t \right\}$;
8          Calculate $L_t$ given $v'$ and $n'$;
9          $dpp \leftarrow D(t) \cdot L_t - V \cdot U_t$;
10         if $dpp < dpp'$ and $L_t \le L_{max}$ then
11             $dpp' \leftarrow dpp$, $v_t \leftarrow v'$, $n_t \leftarrow n'$;
12         Choose another version $v''$ that has not been
             tested, and let $v' \leftarrow v''$;
13     until all versions in $\mathcal{V}$ have been tested;
14     $D(t+1) \leftarrow [D(t) + L_t - L_{max}]^+$;
15     return $v_t$ and $n_t$;
```

Lemma 1: For all possible resulting values of $D(t)$ resulted from any decision over all time slots, the following inequality holds

$$\begin{aligned}&\Delta(D(t)) - V \cdot E[U_t \mid D(t)] \le \\ &B + D(t)E[(L_t - L_{max}) \mid D(t)] - V \cdot E[U_t \mid q(t)],\end{aligned} \tag{6}$$

where we define $L_{up} = \max_{t \in T}\{L_t\}$ indicates the largest delay among all slots, and $B = \frac{1}{2}(L_{up} - L_{\max})^2$ is constant in any time slot. We detail the proof as follows:

Proof. Based on Eq. (4), we drive:

$$\begin{aligned}\Delta(D(t)) &= E[L(D(t+1)) - L(D(t)) \mid D(t)] \\ &\le \frac{1}{2} E\left[(D(t) + L_t - L_{\max})^2 - D(t)^2 \mid D(t) \right] \\ &= \frac{1}{2}(L_t - L_{\max})^2 + D(t)E[(L_t - L_max) \mid D(t)] \\ &\le B + D(t)E[(L_t - L_{\max}) \mid D(t)],\end{aligned}$$

where $B = \frac{1}{2}(L_{up} - L_{\max})^2$ is a constant at any time slot and $l_{up} = \max_{t \in T}\{L_t\}$. Incorporating the expected utility to both sides of Eq. (7), then we have

$$\begin{aligned}&\Delta(D(t)) - V \cdot E[U_t \mid D(t)] \le \\ &B + D(t)E[(L_t - L_{max}) \mid D(t)] - V \cdot E[U_t \mid q(t)]. \qquad \square\end{aligned}$$

Based on **Lemma 1**, we transform problem $\mathcal{P}_1$ into a new real-time optimization problem $\mathcal{P}_2$, which aims to minimize the supremum bound for the DPP function, i.e.,

$$\begin{aligned}\mathcal{P}_2 : \min_{\{v_t, n_t\}} \quad & D(t) \cdot L_t - V \cdot U_t \\ \text{s.t.} \quad & C_1, C_2.\end{aligned} \tag{7}$$

For the term $D(t) \cdot L_t$, it is critical to minimize L_t when $D(t)$ is large. Considering the number of possible solutions (v_t, n_t) that meet the constraints are limited, hence we propose a greedy-based Min-DPP algorithm, which involves iteratively computing $D(t) \cdot L_t$ and $V \cdot U_t$ for each possible (v_t, n_t), and chooses the optimal solution. As calculating

$D(t) \cdot L_t$ and $V \cdot U_t$ are simple computations, Min-DPP algorithm incurs negligible overhead. The details are showed in Algorithm 1.

Theorem 1: The Min-DPP algorithm is able to achieve the performance bound in term of the time-average utility and delay queue backlog as follows:

$$\lim_{T \to +\infty} \frac{1}{T} \sum\nolimits_{t=0}^{T} E\left[U_t\right] \geq \nu^* - \frac{B}{V}, \tag{8}$$

$$\lim_{T \to +\infty} \frac{1}{T} \sum\nolimits_{t=0}^{T} E\left[L_t\right] \leq \frac{B}{\varepsilon} + \frac{V}{\varepsilon}\left(\nu^* - \nu'\right) + L_{\max}, \tag{9}$$

where ν' is the objective value using the worst solution for $\mathcal{P}_1$, ν^* is the achieved optimal utility without considering the delay constraint, and the positive constant ε indicates the obtained long-term delay surplus through several stationary control policies. The detailed proof is as follows:

Proof. To prove the performance guarantees, we first introduce **Lemma 2**, i.e.

Lemma 2: Given any $\delta > 0$, there exists a stationary and randomized strategy Γ for $\mathcal{P}_1$ to decide v_t^Γ and n_t^Γ the independent of the delay queue backlog $D(t)$, such that the following inequalities are satisfied:

$$E\left[L_t^\Gamma - L_{\max}\right] \leq \delta, \text{ and } E\left[U_t^\Gamma\right] \leq \nu_* + \delta,$$

Please refer to Theorem 4.5 in [22] for the proof of the above inequalities, we omit it for brevity.

ViChaser aims to minimize $\mathcal{P}_2$ among the possible decisions. We incorporate **Lemma 2** into Eq. (7), then

$$\begin{aligned} &\Delta(D(t)) - V \cdot E\left[U_t \mid D(t)\right] \\ &\leq B + D(t) E\left[\left(L_t^\Gamma - L_{\max}\right) \mid D(t)\right] - E\left[U_t^\Gamma \mid D(t)\right] V \\ &\leq B + \delta D(t) - V\left(\nu_* + \delta\right). \end{aligned}$$

Summing Eq. (10) over all the time slots and calculating the time-average value, meanwhile letting δ approach 0, we get

$$\frac{1}{T} E[L(D(t)) - L(D(0))] - \frac{V}{T} \sum_{t=0}^{T-1} E\left[U_t^\Gamma\right] \leq B - V \cdot \nu^*.$$

Considering $L(D(t)) \geq 0$ and $L(D(0)) = 0$, hence we rearrange Eq. (10) and get the time-average delay bound

$$\lim_{T \to +\infty} \frac{1}{T} \sum\nolimits_{t=0}^{T} E\left[U_t\right] >= \nu^* - \frac{B}{V}.$$

Assuming there exists a $\varepsilon \leq 0, \Phi(\varepsilon)$ and decision π, s.t.,

$$E\left[L_t^\pi - L_{\max}\right] \leq -\varepsilon, \text{ and } E\left[U_t^\pi\right] = \Phi(\varepsilon).$$

Plugging it into Eq. (7), we get

$$\Delta(D(t)) - V \cdot E\left[U_t\right] \leq B - \varepsilon D(t) - V\Phi(\varepsilon).$$

Summing the above over all slots, and then

$$\begin{aligned} \frac{1}{T} \sum\nolimits_{t=0}^{T-1} E[D(t)] &\leq \frac{\left[B - V\left(\Phi(\varepsilon) - \frac{1}{T}\sum_{t=0}^{T-1} E\left[U_t^\pi\right]\right)\right]}{\varepsilon} \\ &\leq \frac{B}{\varepsilon} - \frac{\left(v^* - v'\right) V}{\varepsilon}. \end{aligned}$$

Considering $\sum_{t=0}^{T-1} E[D(t)] \geq \sum_{t=0}^{T-1} E\left[L_t - L_{\max}\right]$, then

$$\frac{1}{T} \sum\nolimits_{t=0}^{T-1} E\left[L_t\right] \leq \frac{B}{\varepsilon} - \frac{\left(v^* - v'\right) V}{\varepsilon} + L_{\max}.$$

□

Eqs. (8) and (9) characterize the utility-delay trade-off within $[\mathrm{O}(1/V), \mathrm{O}(V)]$. We drive the optimal utility ν^* by adopting an arbitrarily large value of V. However, the time-averaged delay queue backlog grows linearly with V. To sum up, the utility-delay tradeoff allows ViChaser make flexible decisions on the quality version of original video and label quantity.

5 VICHASER MEDIA SERVER DESIGN

5.1 Content-Adaptive Trainer

Generally speaking, the video content (e.g., locations, objects, and even the topics) in the same stream varies over time, hence the pre-trained model probably cannot utilize the video dynamics to further enhance the video quality. In such dilemma, ViChaser adopts an online learning method to exhibits its advantage in quality gain. In specific, for the streaming sessions with frequent scene-changes (e.g., NBA live videos), online learning improves the quality significantly in the beginning, but shows diminished quality gains and finally saturates with the increase of training episodes. On the contrary, if the videos in current streaming session are virtually unchanged (e.g., live Concert video), there is no need to perform persistent online learning due to the substantial redundancy across frames. In fact, the persistent online learning is data- and compute-intensive while the streamer client is resource-constrained, hence we deploy it in the media server and generate training labels from the streamer client, which has been detailed in Section 4.

To measure the quality gain of current model, we define the function $Q_{sr}^{ca}(n_t)$ as the average quality improvement using current online model compared to the pre-trained model when feeding n_t labels at slot t. Once the online learning has been done, the *Content-adaptive Trainer* passes the results to the *Quality Measurer* to profile the latest $Q_{sr}^{ca}(\cdot)$, and return it to the streamer client to perform Min-DPP algorithm for bandwidth allocation. Note that $Q_{sr}^{ca}(\cdot)$ is designed for the specific video category ca, thus once the ca transforms into another category ca', the *Content-adaptive Trainer* will perform retraining based on the initial model, and *Quality Measurer* profiles the new function $Q_{sr}^{ca'}(\cdot)$.

After a certain number of training episodes, online learning reaches a point of diminishing returns in terms of quality improvement. To investigate this saturation, the Content-adaptive Trainer calculates the quality gap at the end of each training epoch by applying the two most recent models to the recent high-quality labels. If the quality gap is lower than a given threshold, a saturation is detected and further label feeding no longer provides significant improvement to online learning. Therefore, the media server logs this saturation, suspends online learning, and send the heartbreak to the streamer client. In turn, the *Transmission Controller* sets n_t in Min-DPP algorithm to a low fixed value (e.g., 5) and selects the resolution v_t that perfectly matches the available bandwidth. It is common for video content, such as locations, objects, and even topics, in the same stream to vary over time. Therefore, a fixed model may not be able to effectively utilize video dynamics to further enhance video quality. However, measuring these variations is challenging. To address this issue, we periodically restart online learning with a time interval of 15 minutes. If the new models do not show significant quality improvements, we suspend online learning.

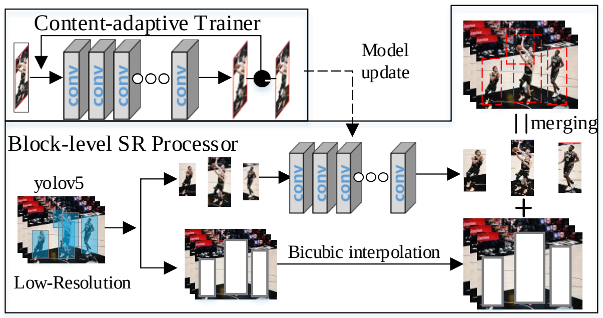


Fig. 5: Workflow of ViChaser in the media server.

5.2 Light detector

In view of the streamer client is resource-constrained and the media server is GPU-enabled, hence we employ powerful DNNs in the media server to identify these potential blocks. Specifically, we use yolov5 [19] algorithm to output all the bounding boxes, each of which contains 5 elements (e.g., X-coordinate, Y-coordinate, length, width, and confidence) and identifies a potential block that users show interests in. To evaluate its detecting rate on heterogeneous devices, we download a NBA game video with duration one hour from YouTube, and employ TensorRT-embedded [20] yolov5 algorithm for detecting in a NVIDIA 2080Ti GPU-provisioned server and a NVIDIA jetson TX2 developer. As Fig. 4a shows, the detecting operation only causes 15 milliseconds in TX2 developer and 6 milliseconds in GPU-provisioned server, which satisfies the real-time requirement of live video streaming with a fps of at least 30.

Note that the detecting results are used for both the server side and the client side. The media server performs block-enhanced super-resolution on this detecting bounding boxes, and the streamer client generates the labels based on the feedback detecting results. Considering the detecting and transmission delay, rather using the results of current frames, the client produces labels according to the previous frames. To mitigate the errors due to both the scene-changes and the different resolutions in the server and client, the client will first enlarge the bounding boxes and then crop these blocks. Specifically, we use (x_1, x_2, y_1, y_2) to denote the coordinates of a block, thus for the bounding box $(21, 32, 10, 20)$ generated from the server using 360p resolution, the client will resize it to $(18, 35, 7, 23)$ using a expansion index of 3, and upscales it to $(54, 105, 21, 69)$ if the resolution of raw live video is 1080p.

5.3 Block-level Super-resolution Processor

ViChaser enhances the quality within the block corresponding to user viewpoint, thus it first obtains the initial blocks based on the detecting results of light detector, then merges the interactive blocks, and concurrently performs DNN super-resolution on these blocks and bi-cubic interpolation on the other areas of the frame with the same upscale factor. Finally, it merges them into a quality-adaptive live video.

5.3.1 *Target block preparation*

It is general that a live video records a kind of group activity such as basketball game or LOL game, hence the end-viewers may pay close attention simultaneously to these interactive objects, which are probably adjacent in the frame. In such case, employing neural enhancement for each block separately may introduce a visual gap between these blocks. A sensible method is to find a sufficiently large area to covers all the involved objects, and directly enhance the final resulting block. Specifically, we use *IoU* (Intersection over Union) metric, which is widely used to evaluate detecting performance. *IoU* is the result of the overlapping area of the two regions (e.g., b_1 and b_2) divided by their union area, i.e., $IoU(b_1, b_2)$ equals $\frac{Area(Overlap(b_1,b_2))}{Area(Union(b_1,b_2))}$, where function $Area(\cdot)$ is to calculate the area. Hence give a threshold η, if $IoU(b_1, b_2)$ is bigger than η, we merge them into a new larger block b_3. Assuming that b_1 and b_2 are (x_1, x_2, y_1, y_2) and (x_3, x_4, y_3, y_4) respectively, then b_3 is $(min(x_1, x_3), max(x_2, x_4), min(y_1, y_3), max(y_2, y_4))$.

In addition to merging the interactive blocks, we propose to filter out the unimportant block. We observe that end-viewers show little interests to two types of objects as follows: (1) objects with small pixel sizes, (2) objects in the background. Hence, we filter out the blocks that containing these objects. Through these two steps, we largely decrease the number of blocks, thereby accelerating the inference.

5.3.2 *Parallel block and frame super-resolution*

As Fig. 5 shows, the *Block-level SR Processor* performs parallel super-resolutions as follows: (1) It utilizes the the latest online model from *Content-adaptive Trainer* to reconstruct the identified blocks of current frame with a given upscale factor; (2) It performs bicubic interpolations on the other areas of the current frame with the same upscale factor.

Having done parallel super-resolution, ViChaser merges the upscaled blocks and frame. Note that applying different upscaling methods on different images and stitching them may incur blocking artifacts. To address this issue, we utilize the deblocking filter tool that is widely used in the mainstream codecs (e.g., HEVC/H.265). Moreover, this tool does not cause much time overhead. Hence, leveraging it does not bring any negative effect in the merging process.

6 Implementation and Evaluation

We have implemented both the streamer client and media server of ViChaser fully based on commodity hardware. Further, we evaluates its performance using multiple types of videos compared to several existing methods.

6.1 Prototype Implementation

6.1.1 *Streamer client*

We implement the streamer client side of ViChaser on a mobile development board NVIDIA Jetson TX2 provisioned with NVIDIA Pascal GPU and Denver CPU, which connects a Hikvision camera to capture the raw live video. For each captured frame, EdgeDiff filter extracts edges using the embedded Canny edge detector in the OpenCV. The streamer client communicates with the media server via TCP connection, which guarantees the orderly transmission of the blocks in the priority label queue.

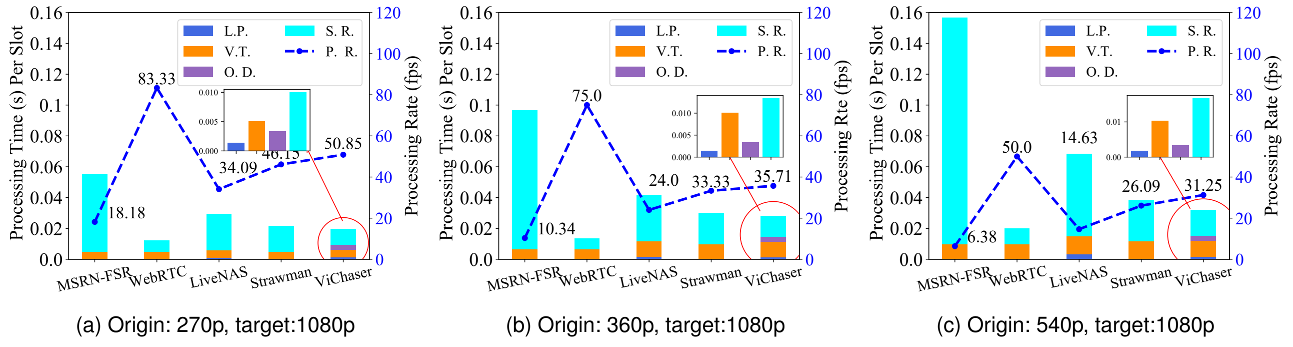


Fig. 6: Average latency measurements when processing different origin resolutions. L. P., V. T., O. D., S. R., and P. R. indicate the labels preparation, video transmission, object detection, super resolution and processing rate, respectively.

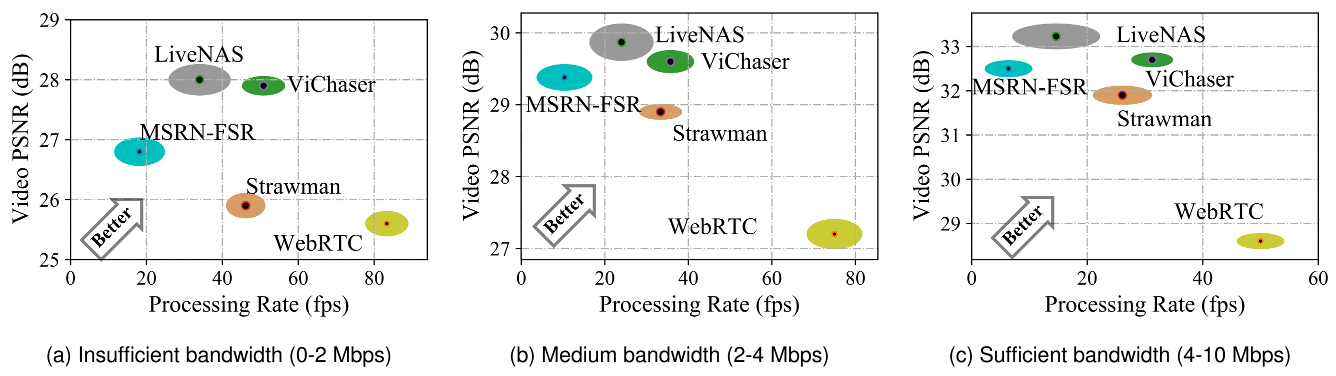


Fig. 7: Video PSNR vs. processing rate under different network conditions.

6.1.2 Media server

We emulate the media server with a server (PowerEdge R740) configured with a NVIDIA GeForce RTX2080 Ti GPU. The media server runs an Ubuntu 16.04 OS, and most of the functions of server side is implemented in Python 3.6.2. Upon receiving the high-quality labels from client, the server performs online learning using PyTorch 1.9.0. ViChaser applies TensorRT yolov5 model to detect the potential blocks of interest, and performs bicubic interpolation provided in the OpenCV.

6.2 Experimental Setup

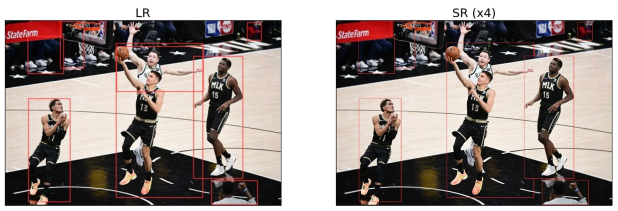
We utilize the model MSRN [14] to perform block-enhanced super-resolution in GPU-provisioned media server. Firstly, we pre-train MSRN DNNs with dataset DIV2K [21], which has a total of 1000 2K resolution images including 800 images for training, 100 images for validation, and 100 images for testing. Note that ViChaser upscales the origin video with low resolution (e.g., 270p, 360p, and 540p) to 1080p, hence ViChaser prepares three pre-trained MSRN models with upscale factor 2, 3 and 4 respectively. Based on the packet reception in the underlying transmission layer, ViChaser obtains the bandwidth estimation, which is used to transmit low-quality videos and high-quality labels.

For the dataset used for training and testing, we collect three categories of live videos from YouTube including the National Basketball Association (NBA) game videos, League of Legends (LOL) game videos, and Live Concert videos. For each category of the above videos, we sample 5 videos with a total duration of about 100 minutes. Note that each sub-video has three resolution versions (e.g., 270p, 360p and 540p), which are available by transcoding from 1080p. We take the widely used PSNR (peak signal to noise ration) as the evaluating metric to measure the video quality.

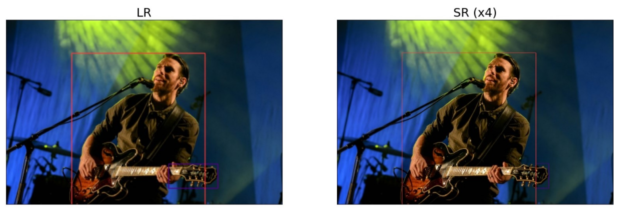
We employ TensorRT-embedded yolov5 as the detector to identify and crop the target blocks. Considering that the detecting classes vary with the types of video (i.e, basketball stars and balls in NBA, but heroes in LoL), one model cannot fit all types of videos. Hence, for each type of videos, we sample substantial pictures and make annotations on them using the tool MakeSense [23], and subsequently train its exclusive detecting model. To verify ViChaser's performance in processing rate and quality improvements, we compare ViChaser with some exiting systems:

- **MSRN-FSR** applies the fixed MSRN model trained with dataset DIV2K [21] to upscale the whole frame.
- **WebRTC** [35] runs bilinear interpolation to get the target resolution version without utilizing DNNs.
- **LiveNAS** [38] employs online learning to maximize the video quality, performs frame-level neural super-resolution when the target resolution is 1080p and supports multi-GPU slice-parallel inference to generate ultra-high resolution videos (e.g., 4k).
- **Strawman** identifies the target blocks based on the *Canny Detector* used in Supremo [39], and performs super-resolution using MSRN models without online learning.

Note that MSRN-FSR, WebRTC and Strawman do not consider the online learning to update the model. LiveNAS is proposed to generate 4k video by utilizing slice-parallel super-resolution using multiple servers. For fairness, only one RTX2080 Ti GPU is available. Strawman identifies its blocks by specific tools in the OpenCV.

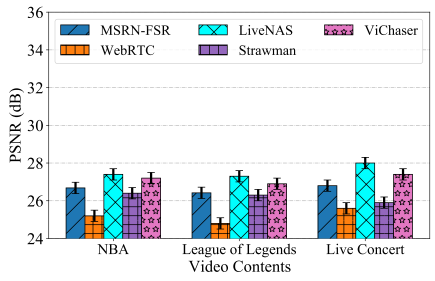


(a) Block-enhanced super-resolution for NBA live video

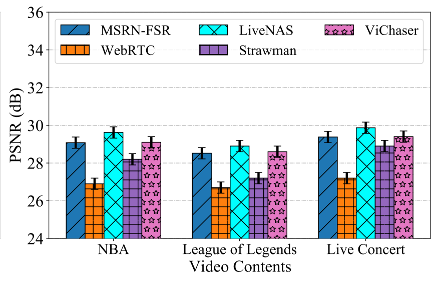


(b) Block-enhanced super-resolution for live concert video

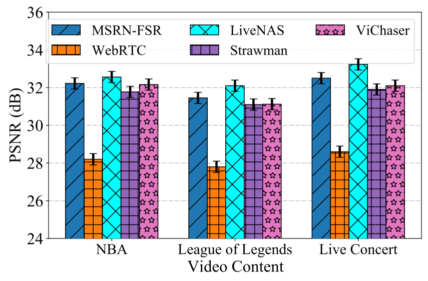
Fig. 8: Block-enhanced super-resolution.



(a) Origin: 270p, target:1080p



(b) Origin: 360p, target:1080p



(c) Origin: 540p, target:1080p

Fig. 9: Video quality measurements when ViChaser upscales the origin version to the target version.

6.3 Ingest Acceleration of ViChaser

Live video streaming application is delay-sensitive, hence it is critical to reduce the time overhead generated from video transmission and super-resolution in media server side. We evaluate ViChaser's end-to-end delay (i.e., label preparation time, detection time, transmission time, and inference time) compared to the listed designs as Fig. 6 shows. We first analyze the delay shares of each processing component. Regardless of the original resolution version, the super-resolution delay takes up the most of the total processing time, especially when the origin resolution is 540p, which further demonstrates that performing super-resolution for videos with large input size is expensive. MSRN-FSR upscales the video resolution by means of frame-enhanced super-resolution, which incurs a unbearable inference delay. WebRTC executes super-resolution through simple bilinear interpolation, which causes little delay. LiveNAS decreases the DNN inference delay via computations distribution for each GPU core, but only one RTX2080 Ti GPU is provisioned, hence LiveNAS still generate large amount of inference delay. Strawman and ViChaser mitigate DNN inference by reducing the origin input size for super-resolution, but the block size of Strawman is larger than that in ViChaser, thus that Strawman introduces more inference delay. Despite ViChaser employs the yolov5 detector, it incurs negligible detecting delay which is about 6 milliseconds.

Along with the increase of original resolution, the total processing time is significantly increasing. For example, ViChaser takes 43% and 65% more time when the original resolution changes from 270p to 360p, and 270p to 540p respectively, which is in accord with our findings in Section 2.

Based on the resulting processing time, we calculate the processing rate. In general, video encoded with a fps of at least 16 is considered to be fluent. Due to the expensive inference delay, MSRN-FSR could only achieve 18, 10 and 6 fps processing rate when the original resolution is 270p, 360p and 540p respectively. On the contrary, without running DNN inference, interpolation-based WebRTC supports live video streaming. When upscaling the 540p video to 1080p video, LiveNAS fails to provide real-time high quality video, which is hindered by the limited available GPU resources. Strawman and ViChaser obtain the fine processing rate that perfectly matches the real-time feature of live streaming. Compared to MSRN-FSR, LiveNAS and Strawman, ViChaser achieves 25, 16, 5 higher fps respectively when original resolution is 540p, and 26, 11, 2 higher fps respectively when original resolution is 360p.

6.4 Video Quality Measurement

Apart from the video fluency, video quality significantly affects user's quality of experience. We first present the visual effect of block-enhanced super-resolution in Fig. 8. For example in a live NBA game, we enhance the quality within the blocks that contain players and balls. For the music concert, we improve the quality of the blocks consist of singers. From the perspective of users, their perceived quality is hardly be degraded because their blocks of interest are high-quality provided. Then to quantify the quality improvement, we depict the achieved PSNR of ViChaser and other designs when applying them to upscale the origin resolution to target resolution as Fig. 9 shows. WebRTC achieves a low quality due to the severe hidden information loss when directly running bilinear interpolation. Compared to MSRN-FSR with pre-trained model, LiveNAS and ViChaser using persistently trained model delivers robust quality gain due to the independence of training data set and the live video. ViChaser outperforms MSRN-FSR by 0.1 to 0.4 dB in PSNR, and Strawman by 0.4 to 1.5 dB, owing to the persistent online learning. LiveNAS achieves higher quality compared to ViChaser, which is mainly benefited from the frame-enhanced or slice-parallel super-resolution that incurs a prohibitive delay. Meanwhile, upscaling a lower origin version to the same target version leads a lower video quality. For example, the achieved quality of ViChaser is decreased by 1.9 dB when the input resolution changes from 540p to 270p.

In addition, we find that the video quality varies with the video content. For example when upscaling 360p to 1080p, the reconstructed videos of Live Concert achieve the highest quality. Considering the videos of Live Concert incurs fewer scene changes, hence the pre-trained super-resolution DNN still works, and the online learning gain shows diminishing return rapidly and gets saturated. However, the other two types of videos appear to have frequent scene changes, hence the pre-trained model does not fit the video content, and persistent online learning is required to enhance the video quality. This finding motivates us that to adaptively adjust the bandwidth allocation for the transmission controller in Section 4.3. When the video has frequently scene-changes, the transmission controller will lower the video resolution, and increase the amount of high-quality labels.

Note that ViChaser aims to accelerate the reconstruction process, thereby meeting the real-time feature but not degrading user's perceived quality. We calculate the PSNR in the whole frame, hence the frame-enhanced or slice-parallel LiveNAS gains a slightly higher PSNR than ViChaser. However, ViChaser maintains a high-quality within the blocks users show interests in, hence LiveNAS and ViChaser get almost the same perceived quality. What's more, ViChaser owns a much higher processing rate than LiveNAS, which is the key contribution of this paper.

6.5 Performance under different network conditions

In order to compare the performance of ViChaser with other designs under different network conditions, we present three levels of available bandwidth, namely insufficient, medium, and sufficient, corresponding to 0-2, 2-4, and 4-10 Mbps, respectively. The resulting video Peak Signal-to-Noise Ratio (PSNR) and processing rate are illustrated in Figure 7. As shown in Fig.7a, when the bandwidth is insufficient, ViChaser achieves a processing rate of 52.1 fps, which outperforms LiveNAS (34 fps) and MSRN-FSR (18.5 fps) by 53.2% and 181.6%, respectively. Despite a slight PSNR drop (0.1 dB), ViChaser demonstrates the best overall performance. While WebRTC attains high processing rates, it exhibits poor video quality. With a medium bandwidth, as depicted in Fig.7b, ViChaser improves processing rate by 34.71% compared to LiveNAS. In reliable network conditions with sufficient bandwidth, as shown in Fig. 7c, ViChaser still achieves a processing rate of 32 fps, whereas LiveNAS can only achieve 15 fps. It should be noted that, despite the PSNR improvements resulting from sufficient bandwidth, the larger resolution of the original video incurs higher server-side reconstruction time.

6.6 Bandwidth Usage and GPU Utilization

We also monitor the bandwidth usage for transmitting low quality video and training labels, and the GPU utilization for block-enhanced super-resolution and online learning. As Fig. 10a shows, ViChaser allocates more bandwidth to transmit high quality labels in the beginning, but reduces the share over time, and finally allocates the majority of available bandwidth to transmit low quality video. As stated before, for a new type of live video, online learning improves the quality a lot, hence the labels have a higher priority, but online learning shows diminishing return when it incurs few scene-changes over time; hence, to further enhance the video quality, ViChaser prefers to transmit video with quality as high as possible. In addition, we test the bandwidth share for live videos by setting different pairs (ω_1, ω_2). It is clear that increasing ω_1 makes ViChaser allocating more bandwidth for live videos to obtain a higher quality gain. When ω_1 equals 1, the online learning is suspended, and all the bandwidth is used for transmitting live videos. We also depict the GPU utilization of ViChaser as Fig. 10b shows. For the NBA video, ViChaser only utilizes 38.2% GPU resources when the original resolution is 270p, but it reaches 75.77% when given 540p. In addition, the videos with few scene-changes such live concert video utilize less GPU resources because it costs less to perform online learning; but ViChaser needs to run persistent online learning for NBA video.

7 Discussion

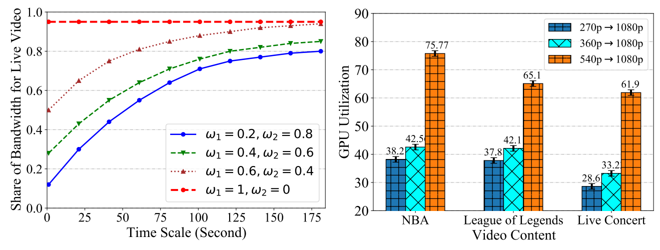
ViChaser uses a server-driven method to predict the generate the high-quality labels in the streamer side, however, probably causing extra network round-trip times (RRTs) to receive the feedback. What's more, despite the availability of videos with all quality levels, users can not enjoy the high-quality videos when encountering terrible downlink network status. ViChaser does not consider to solve the downlink network fluctuations.

Moving forward, with the emerging of mobile edge computing, the mobile end-devices are provisioned with more and more powerful computing resource, which enable deep neural networks to be ran locally. Based on this, it is promising to deploy the YOLOv5 models in the streamer side to reduce the RRTs. In addition, we try to migrant neural super resolution from media server to user end-devices, which probably alleviate the downlink network fluctuations.

8 Related Work

Adaptive streaming. DASH (Dynamic Adaptive Streaming over HTTP) is the key mechanism to enhance bandwidth utilization. By integrating ABR (Adaptive Bitrate) algorithms on the client side, DASH selects the adaptive bitrate that best matches the available bandwidth and the buffer occupancy level of video players. This solutions consists of three categories, e.g. rate-based algorithm [34], [35], [36], buffer-based algorithms [31], [32], [33], and combination of the above algorithms [29], [30]. MPC [29] chooses the best bitrate by optimizing the maximum user's QoE based on the bandwidth and buffer occupancy. Pensieve [29] selects the adaptive bitrate via reinforcement learning. Though efficiently utilizing bandwidth or mitigate buffer occupancy, these schemes face the same challenge when the available bandwidth is insufficient, these schemes are not capable of providing high quality video for users. In general, they upscaled the origin version to a high version through simple interpolation method such as bilinear interpolation in WebRTC [35]. ViChaser achieves a high bandwidth utilization, and enables media server to generate high quality video by utilizing DNN super-resolution.

Deep super-resolution (SR). SRCNN [24] first proposed to employ DNN to perform SR on single image to reconstruct high-resolution image. Since then, further improved



(a) Share of bandwidth (b) GPU utilization

Fig. 10: Bandwidth and GPU usage.

model architectures [14], [16], [17] have been actively studied. By considering temporal consistency, Video SR [25], [26], [27], [28] extends and develops the single image SR, and scales up the low resolution to target resolution. More and more attractions focused on enhancing the quality of video streaming. For example, NAS [37] provides multiple models for users to enhance the video quality in mobile device, LiveNAS [38] aims to resolve the uplink bandwidth variation and employ online learning to maximize the quality gain, Supremo [39] offloads partial blocks to server for super resolution without content-adaptive module, and PARSEC [40] enhances 360-degree video streaming using super-resolution. However, NAS and LiveNAS rely largely on the GPU provision, Supremo and PARSEC is hardly generalized to other types of video content.

9 Conclusion

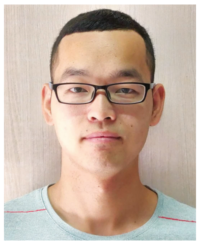
In this paper, we present ViChaser, an innovative live streaming system that applies super-resolution into the potential blocks to enhance user's perceived video quality without depending on the uplink bandwidth. ViChaser adopts online learning method to adapt to the content of live video, and crops the corresponding blocks as training labels based on the detection feedback in the media server. To efficiently utilize the limited uplink bandwidth, we leverage Lyapunov framework to allocate the bandwidth by controlling the origin video resolution and the labels quantity. ViChaser merges the DNN upscaled blocks and upscaled videos into the block-adaptive live videos. The results show that ViChaser achieves 1.2-1.5 dB higher video quality over WebRTC, and increases the processing speed by 11-16 fps compared with LiveNAS.

Nonetheless, ViChaser left many problems to be solved as follows: 1) applying yolov5 to the videos with low resolution may lead to a low detecting accuracy, which affects the final perceived video quality; 2) there exists overlay area between the detected blocks, and how to integrate these blocks is a critical challenge; 3) can ViChaser outperform other schemes when given sufficient GPU resources? We will present our solution in our future work.

References

[1] Youtube Live Streaming Official Website. https://www.youtube.com/live.
[2] Twitch Official Website. https://www.twitch.tv/.
[3] Cisco Annual Internet Report. https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html.
[4] RTSP. https://tools.ietf.org/html/rfc2326.
[5] RTP. https://tools.ietf.org/html/rfc3550.
[6] DASH Industry Forum Official Website. https://dashif.org/.
[7] R. Netravali, A. Sivaraman, S. Das, A. Goyal, K. Winstein, J. Mickens, and H. Balakrishnan. 2015. Mahimahi: Accurate record-and-replay for HTTP. In *USENIX ATC*, 417–429.
[8] Y. Sun, X. Q. Yin, J. C. Jiang, V. Sekar, F. Y. Lin, N. S. Wang, T. Liu, and B. Sinopoli. 2016. CS2P: Improving video bitrate selection and adaptation with data-driven throughput prediction. In *ACM SIGCOMM*, 272–285.
[9] J. C. Yang, J. Wright, and T. Huang. 2008. Image super-resolution as sparse representation of raw image patches. In *IEEE CVPR*, 1–8.
[10] R. Timofte, V. D. Smet, and L. V. Gool. 2014. A+: Adjusted anchored neighborhood regression for fast super-resolution. In *Springer ACCV*, 111–126.
[11] S. Schulter, C. Leistner, and H. Bischof. 2015. Fast and accurate image upscaling with super-resolution forests. In *IEEE CVPR*, 3791–3799.
[12] B. Lim, S. Son, H. Kim, S. Nah, and K. M. Lee. 2017. Enhanced deep residual networks for single image super-resolution. In *IEEE CVPR*, 136–144.
[13] Y. L. Zhang, K.P. Li, K. Li, L. C. Wang, B. Zhong, and Y. Fu. 2018. Image super-resolution using very deep residual channel attention networks. In *Springer ECCV*, 286–301.
[14] N. Ahn, B. Kang, and K. A. Sohn. 2018. Fast, accurate, and lightweight super-resolution with cascading residual network. In *Springer ECCV*, 252–268.
[15] J.H. Kim and J.S. Lee. 2018. Deep residual network with enhanced upscaling module for super-resolution. In *IEEE CVPR*, 913–921.
[16] J. C. Li, F. M. Fang, and K. F. Mei. 2018. Multiscale residual network for image super-resolution. In *Springer ECCV*, 517–532.
[17] Y. Guo, J. Chen, J. D. Wang, Q. Chen, J. Z. Cao, Z. S. Deng, Y. Xu, and M. Tan. 2020. Closed-loop matters: Dual regression networks for single image super-resolution. In *IEEE CVPR*, 5407–5416.
[18] The Berkeley Segmentation Dataset and Benchmark. https://www2.eecs.berkeley.edu/Research/Projects/CS/vision/bsds/.
[19] yolov5. https://github.com/ultralytics/yolov5.
[20] TensorRT. https://developer.nvidia.com/tensorrt.
[21] DIVerse 2K resolution high quality images as used for the challenges. https://data.vision.ee.ethz.ch/cvl/DIV2K/.
[22] Michael J Neely. 2010. Stochastic network optimization with application to communication and queueing systems. In *Synthesis Lectures on Communication Networks*, 1–211.
[23] MakeSense. https://www.makesense.ai/.
[24] C. Dong, C. Loy, and K. He. 2014. Learning a deep convolutional network for image super-resolution. In *Springer ECCV*, 184–199.
[25] Y. Jing, Y. Yang, X. Wang, M. Song, and D. Tao. 2021. Turning Frequency to Resolution: Video Super-Resolution via Event Cameras. In *IEEE CVPR*, 7772–7781.
[26] Y. Jo, S. W. Oh, J. Kang, and S. J. Kim. 2018. Deep video super-resolution network using dynamic upsampling filters without explicit motion compensation. In *IEEE CVPR*, 3224–3232.
[27] J. Caballero, C. Ledig, A. Aitken, A. Acosta, J. Totz, Z. Wang, and W. Shi. 2017. Real-time video super-resolution with spatio-temporal networks and motion compensation. In *IEEE CVPR*, 4778–4787.
[28] Y. Zhang, Y. Zhang, Y. Wu, Y. Tao, K. Bian, P. Zhou, L. Song, and H. Tuo. 2020. Improving quality of experience by adaptive video streaming with super-resolution. In *IEEE INFOCOM*, 1957–1966.
[29] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli. 2015. A control-theoretic approach for dynamic adaptive video streaming over HTTP. In *ACM SIGCOMM*, 325–338.
[30] H. Mao, R. Netravali, and M. Alizadeh. 2017. Neural adaptive video streaming with pensieve. In *ACM SIGCOMM*, 197–210.
[31] K. Spiteri, R. Urgaonkar, and R. K Sitaraman. 2020. BOLA: Near-optimal bitrate adaptation for online videos. In IEEE/ACM Transactions on Networking 28, 4 (2020), 1698–1711.
[32] T. Huang, R. Johari, N. McKeown, M. Trunnell, and M. Watson. 2014. A buffer-based approach to rate adaptation: Evidence from a large video streaming service. In *ACM SIGCOMM*, 187–198.
[33] K. Spiteri, R. Sitaraman, and D. Sparacio. 2019. From theory to practice: Improving bitrate adaptation in the DASH reference player. In ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM) 15, 2s (2019), 1–29.
[34] H. Yuan, X. Hu, J. Hou, X. Wei, and S. Kwong. 2019. An ensemble rate adaptation framework for dynamic adaptive streaming over HTTP. In IEEE Transactions on Broadcasting 66, 2 (2019), 251–263.
[35] G. Carlucci, L. D. Cicco, S. Holmer, and S. Mascolo. 2016. Analysis and design of the google congestion control for web real-time communication (WebRTC). In *ACM MM*, 1–12.

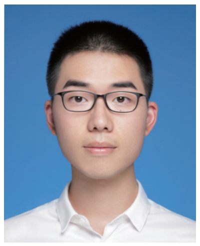
[36] J. Jiang, V. Sekar, and H. Zhang. 2012. Improving fairness, efficiency, and stability in http-based adaptive video streaming with festive. In *CoNEXT*, 97–108.
[37] H. Yeo, Y. Jung, J. Kim, J. Shin, and D. Han. 2018. Neural adaptive content-aware internet video delivery. In *USENIX OSDI*, 645–661.
[38] J. Kim, Y. Jung, H. Yeo, J. Ye, and D. Han. 2020. Neural-enhanced live streaming: Improving live video ingest via online learning. In *ACM SIGCOMM*, 107–125.
[39] J. Yi, S. Kim, J. Kim, and S. Choi. 2020. Supremo: Cloud-Assisted Low-Latency Super-Resolution in Mobile Devices. In *IEEE Transactions on Mobile Computing*, 1-13.
[40] M. Dasari, A. Bhattacharya, S. Vargas, P. Sahu, A. Balasubramanian, and S. R. Das. 2020. Streaming 360-degree videos using super-resolution. In *IEEE INFOCOM*, 1977–1986.
[41] J. Kim, J. K. Lee, and K. Mu Lee. 2016. Accurate image super-resolution using very deep convolutional networks. In *IEEE CVPR*, 1646–1654.
[42] W. Shi, J. Caballero, F. Huszár, J. Totz, A. P. Aitken, R. Bishop, D. Rueckert, and Z. Wang. 2016. Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. In *IEEE CVPR*, 1874–1883.
[43] Y. Zhang, Y. Tian, Y. Kong, ans B. Zhong. 2018. Residual dense network for image super-resolution. In *IEEE CVPR*, 2472–2481.
[44] R. Lee, S. I. Venieris, L. Dudziak, S. Bhattacharya, and N. D. Lane. 2019. Mobisr: Efficient on-device super-resolution through heterogeneous mobile processors. In *ACM MOBICOM*. 1–16.
[45] A. Shocher, N. Cohen, and M. Irani. 2018. "zero-shot" super-resolution using deep internal learning. In *IEEE CVPR*, 3118–3126.
[46] B. Gary and K. Adrian. 2008. Learning OpenCV: Computer vision with the OpenCV library. O'Reilly Media, Inc.



Ning Chen is currently pursuing the PhD degree in the Department of Computer Science and Technology, Nanjing University, under the supervision of Prof. Sheng Zhang. His research interests including edge computing, deep reinforcement learning, and video streaming. To date, he has published several papers, including those appeared in INFOCOM, TPDS, TON, SECON, Computer Network, ICPADS, et al.

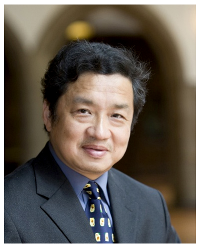
Sheng Zhang is an associate professor in the Department of Computer Science and Technology, Nanjing University. He is also a member of the State Key Lab. for Novel Software Technology. He received the BS and PhD degrees from Nanjing University in 2008 and 2014, respectively. His research interests include cloud computing and edge computing. To date, he has published more than 80 papers, including those appeared in TMC, TON, TPDS, TC, MobiHoc, ICDCS, and INFOCOM. He received the Best Paper Award of IEEE ICCCN 2020 and the Best Paper Runner-Up Award of IEEE MASS 2012. He is the recipient of the 2015 ACM China Doctoral Dissertation Nomination Award. He is a member of the IEEE and a senior member of the CCF.

Zhi Ma received the B.S. degree from the Department of Computer Science and Technology, Nanjing University, in 2017. He is currently working towards the PhD degree with the Department of Computer Science and Technology, Nanjing University, under the supervision of Prof. Sheng Zhang. His research interests include wireles charging and edge computing.



Yu Chen received the BS degree from the Department of Computer Science and Technology, Nanjing University, China, in 2019, where he is currently working toward the PhD degree under the supervision of associate professor Sheng Zhang. He is a member of the State Key Laboratory for Novel Software Technology. Currently, his research interests include mobile edge computing, video analytics and game theory. He has published 4 papers in referred journals and conferences including TPDS, INFOCOM, et al.

Yibo Jin received the BS degree from the Department of Computer Science and Technology, Nanjing University in 2017, where he is currently pursuing the PhD degree under the supervision of Professor Sanglu Lu. He was a visiting student with the Hong Kong Polytechnic University, Hong Kong in 2017. His research interests include big data analytics, edge computing and distributed learning. He is a student member of the IEEE.



Jie Wu (F'09) is the Director of the Center for Networked Computing and Laura H. Carnell professor at Temple University. He also serves as the Director of International Affairs at College of Science and Technology. He served as Chair of Department of Computer and Information Sciences from the summer of 2009 to the summer of 2016 and Associate Vice Provost for International Affairs from the fall of 2015 to the summer of 2017. Prior to joining Temple University, he was a program director at the National Science Foundation and was a distinguished professor at Florida Atlantic University. His current research interests include mobile computing and wireless networks, routing protocols, cloud and green computing, network trust and security, and social network applications. Dr. Wu regularly publishes in scholarly journals, conference proceedings, and books. He serves on several editorial boards, including IEEE Transactions on Mobile Computing, IEEE Transactions on Service Computing, Journal of Parallel and Distributed Computing, and Journal of Computer Science and Technology. Dr. Wu was general co-chair for IEEE MASS 2006, IEEE IPDPS 2008, IEEE ICDCS 2013, ACM MobiHoc 2014, ICPP 2016, and IEEE CNS 2016, as well as program co-chair for IEEE INFOCOM 2011 and CCF CNCC 2013. He was an IEEE Computer Society Distinguished Visitor, ACM Distinguished Speaker, and chair for the IEEE Technical Committee on Distributed Processing (TCDP). Dr. Wu is a CCF Distinguished Speaker and a Fellow of the IEEE. He is the recipient of the 2011 China Computer Federation (CCF) Overseas Outstanding Achievement Award.

Zhuzhong Qian is a professor at the Department of Computer Science and Technology, Nanjing University. He is also a member of the State Key Laboratory for Novel Software Technology. He received his PhD. Degree in computer science in 2007. Currently, his research interests include cloud computing, distributed systems, and pervasive computing. He is the chief member of several national research projects on cloud computing and pervasive computing. He has published more than 30 research papers in related fields. He is a member of CCF and IEEE.

Yu Liang is a lecturer at Nanjing Normal University. She received the MS and PhD degrees from Nanjing University in 2011 and 2021, respectively. She was a senior software engineer in Trend Micro China Development Center between 2011 and 2017. Her research interests include edge intelligence and edge computing. Her publications include those appeared in TMC, TPDS, TON, Computer Networks, Computer Communications, IEEE ICDCS, IEEE MSN, and IEEE Globecom.

Sanglu Lu received her BS, MS, and PhD degrees from Nanjing University in 1992, 1995, and 1997, respectively, all in computer science. She is currently a professor in the Department of Computer Science and Technology and the State Key Laboratory for Novel Software Technology. Her research interests include distributed computing, wireless networks, and pervasive computing. She has published over 80 papers in referred journals and conferences in the above areas. She is a member of IEEE.