EVASR: Edge-Based Salience-Aware Super-Resolution for Enhanced Video Quality and Power Efficiency

NA LI, Rutgers University, USA
ZICHEN ZHU, Rutgers University, USA
SHENG WEI, Rutgers University, USA
YAO LIU, Rutgers University, USA

With the rapid growth of video content consumption, it is important to deliver high-quality streaming videos to users even under limited available network bandwidth. In this paper, we propose EVASR, a system that performs edge-based video delivery to clients with salience-aware super-resolution. We select patches with higher saliency score to perform super-resolution while applying the simple yet efficient bicubic interpolation for the remaining patches in the same video frame. To efficiently use the computation resources available at the edge server, we introduce a new metric called "saliency visual quality" and formulate patch selection as an optimization problem to achieve the best performance when an edge server is serving multiple users. We implement EVASR based on the FFmpeg framework and deploy it on three different platforms including desktop/laptop computers, mobile phones, and single board computers (SBCs). We conduct extensive experiments for evaluating the visual quality, super-resolution speed, and power savings that can be achieved by EVASR. Results show that EVASR outperforms baseline approaches in both resource efficiency and visual quality metrics including PSNR, saliency visual quality (SVQ), and VMAF. EVASR can also achieve substantial energy savings compared to baseline approaches `MobileSR` and `JetsonSR` on mobile devices.

CCS Concepts: • **Information systems** → **Multimedia streaming**; • **Computing methodologies** → **Reconstruction**; **Image processing**.

Additional Key Words and Phrases: Video delivery, edge computing, super-resolution, deep-learning, salience-aware, visual quality

ACM Reference Format:
Na Li, Zichen Zhu, Sheng Wei, and Yao Liu. 2025. EVASR: Edge-Based Salience-Aware Super-Resolution for Enhanced Video Quality and Power Efficiency. *ACM Trans. Multimedia Comput. Commun. Appl.* 1, 1, Article 1 (January 2025), 24 pages. https://doi.org/10.1145/3711928

1 INTRODUCTION

With the increased popularity of video sharing and streaming platforms, nowadays, users are spending more time watching online streaming videos using their devices. However, the limited available network bandwidth cannot keep up with user's ever-increasing demand for higher video quality. To address this problem, advanced video codecs such as VVC [1] and AV1 [2] are developed with the goal of improving the video compression performance. In addition, existing works have also proposed to use super-resolution for trading increased computation for reduced network bandwidth usage and better visual quality, e.g., NAS [3].

With super-resolution, during streaming, a low resolution video is downloaded along with a trained deep-learning model. The size of the model is much smaller compared to the video, e.g.,

Authors' addresses: Na Li, Rutgers University, Piscataway, NJ, USA, na.li@rutgers.edu; Zichen Zhu, Rutgers University, Piscataway, NJ, USA, zichen.zhu@rutgers.edu; Sheng Wei, Rutgers University, Piscataway, NJ, USA, sheng.wei@rutgers.edu; Yao Liu, Rutgers University, Piscataway, NJ, USA, yao.liu@rutgers.edu.

100 KBytes only. At the video receiver, the model is used for upscaling the low resolution frame to higher resolution and enhancing the video quality. For example, NAS [3] uses multi-scale deep super-resolution (MDSR) [4] for upscaling low resolution videos downloaded via DASH to a high resolution of 1080p. To improve the super-resolution visual quality, it proposes to train a model for each video. PARSEC [5] further proposes to reduce the granularity of super-resolution models to one micro-model per video segment. Unlike video compression that aims to optimize the rate-distortion curve, deep-learning-based super-resolution can use additional computation at the video receiver for improving the visual quality. Thus, deep-learning-based super-resolution complements recent advances in next-generation video codecs such as VVC and AV1.

However, an issue with NAS [3] is the slow super-resolution inference speed. It was shown in [6] that the inference throughput of NAS is less than 15 frames per second (fps), which is much smaller than typical video frame rate of 24-30 fps. To enable real-time video quality enhancement, both NEMO [7] and dcSR [6] are proposed to improve the inference speed by applying deep-learning-based super-resolution on selected frames only. To do so, both works require changes to the decoder workflow of the video codec, e.g., the H.264 decoder.

In this work, we propose EVASR, an edge-based video delivery system that uses salience-aware super-resolution to enhance the video quality for clients. It uses the resources at the nearby edge servers for performing computation-intensive super-resolution (Section 3). EVASR is different from prior works in three ways. *First*, we consider a provider-edge-client scenario, where an edge server with computation resources can perform intensive super-resolution tasks on behalf of the client. The edge server may simultaneously serve multiple users, which makes it necessary to allocate resources appropriately among users. *Second*, to efficiently use the limited available computation resources while supporting real-time video super-resolution, we propose to perform deep-learning-based super-resolution only on a selected set of patches in a frame, taking into account different saliency in different parts of the video frame, instead of the full frame. For the remaining patches, EVASR performs bicubic interpolation for upscaling. *Third*, EVASR performs super-resolution directly on decoded frames. It does not require any changes to the video codecs and thus can work with any existing video codecs.

To provide better visual quality among all request users, we formulate the patch selection problem as an optimization problem with the objective of maximizing the saliency visual quality for all users served by the edge server (Section 4). According to Borji and Ltti [8], human eyes tend to focus on attractive and interesting regions or objects, known as salient areas. Inspired by this visual perception phenomenon, we propose a new metric, the saliency visual quality (SVQ), which places more emphasis (i.e., weights) on the visual quality of areas on the frame where human eyes are more sensitive. The goal of our SVQ optimizer (SVQO) is to maximize the saliency visual quality subject to a number of constraints.

We implement our EVASR system based on the FFmpeg framework [9] and deploy it for use with three different types of client devices, laptop/desktop computers, mobile phone, single board computer (SBC) (Section 5). We perform extensive experiments to evaluate the visual quality, super-resolution speed, and the power savings that can be produced by EVASR. Results show that the SVQO optimizer can effectively select a set of patches for super-resolution that improves both the saliency visual quality and the VMAF quality metric. With the SVQO optimizer, EVASR can support real-time per-frame super-resolution to 1920x1080 and can support multiple clients' super-resolution requests simultaneously. For comparison, we also implemented two baseline approaches that perform super resolution on a mobile phone directly (the `MobileSR` approach) and on a Nvidia Jetson Nano directly (the `JetsonSR` approach). Results show that with EVASR, the energy consumption of these mobile devices during super-resolution video streaming is significantly reduced compared to the baseline approaches.

An earlier version of this work was published at MMSys'2023 [10]. In this manuscript, we explore an additional benefit of EVASR for performing power-efficient super-resolution during video streaming on mobile devices. We describe the implementation details for baseline approaches, `MobileSR`, `JetsonSR`, as well as the detailed power measurement setups. In addition, we present detailed evaluation results comparing the performance of EVASR with these baseline approaches on mobile devices.

In summary, this paper makes the following contributions:

- We propose EVASR that performs edge-based video delivery with salience-aware super-resolution with three components: the video service provider, the edge server, and the clients.
- We integrate our EVASR system with super-resolution and saliency detection to achieve patch-based super-resolution on the edge server side. This approach achieves a balance between visual quality performance and computational resource usage.
- To select the set of patches for super-resolution, we formulate the patch selection problem as an optimization problem with the objective of maximizing the saliency visual quality.
- To perform a comprehensive evaluation of our EVASR system, we conducted extensive experiments using high quality videos. Multiple visual quality metrics were used in our evaluation: PSNR, saliency visual quality (SVQ), and VMAF. Results show that EVASR outperforms other baseline approaches in visual quality and energy efficiency.

2 BACKGROUND AND RELATED WORK

2.1 Super Resolution

The super-resolution (SR) field has advanced rapidly from its origins in the deep learning age. The SRCNN model [11, 12] was the first to apply convolutional neural networks (CNNs) to super-resolution. FSRCNN [13] was an evolution of SRCNN. It operated directly on a low-resolution input image and applied a deconvolution layer to generate high-resolution output. VDSR [14] was the first to apply residual layers [15] to the SR task, allowing for deeper super-resolution networks. DRCN [16] introduced recursive learning in a very deep network for parameter sharing. Notably, Shi et al. proposed ESPCN [17] that uses "PixelShuffle", a method for mapping values at low-resolution positions directly to positions in a higher-resolution image. The "PixelShuffle" operation is more efficient compared to the deconvolution operation. Many subsequent super-resolution works use the "PixelShuffle" method in their proposed networks. SRResNet [18] used a modified residual layer tailored for the super-resolution application. EDSR [4] further modified the SR-specific residual layer from SSResNet and introduced a multi-task objective in a new multi-scale deep super-resolution (MDSR) model. SRGAN [18] applied a generative adversarial network (GAN) [19] to SR, allowing better resolution of high-frequency details. More recently, Zhang et al. proposed RCAN [20] that uses a residual in residual (RIR) structure to train very deep neural networks and a channel attention mechanism that considers inter-dependencies among feature channels. RCAN and its improved version, RCAN-it [21], were shown to achieve improved the accuracy and better visual quality in super-resolution tasks. SwinIR [22] is an image restoration model based on the Swin Transformer [23]. When SwinIR is used for x4 super-resolution tasks, it is shown to achieve even better results compared to RCAN.

While recent SR models can achieve good visual quality performances, they are usually very big, containing millions of parameters. As a result, super-resolution inference can take a long time, which makes them infeasible for video-based super-resolution with time constraints. For example, SwinIR contains 11.8 million parameters and super-resolution on a 1024x1024 image takes 1.1 seconds [22]. On the other hand, the ESPCN model has only about 26K parameters while achieving

good visual quality performance. Considering the trade-off between performance and inference speed, we use ESPCN in our current EVASR implementation.

2.2 Video Quality Enhancement

Besides its use in traditional computer vision tasks, recently, super-resolution is also used for video quality enhancement. Due to the limitation of network capacity, it is infeasible for many users to obtain high-quality videos during streaming. Thus, many recent works have been proposed to stream low-resolution videos to end users and enhance the video quality via super-resolution.

For example, NAS [3] is the first work that proposes to perform super-resolution on the client side. It adopts a multi-scale deep super-resolution model MDSR [4] to upscale input videos of different resolutions, e.g., 240p, 360p, 540p, to a 1080p output. NEMO [7] is a system that aims to support real-time video super-resolution on mobile devices from 240p to 960p. Instead of per-frame super-resolution, NEMO only applies super-resolution to a few selected frames, caches their outputs, and transfers them to other frames. However, all frames are important in a video since all of them will be watched by users. If there exists a large quality gap between two consecutive frames, human eyes can be very sensitive to such changes. In addition, caching may also introduce other issues, such as memory. LiveNAS [24] focuses on video uploading and introduces a live video ingest framework that enhances video quality by leveraging the super-resolution model training at ingest servers. It allocates some bandwidth to transmit selected high-quality patches to perform online model training and uses the remaining bandwidth for uploading live video in lower quality. Baek et al. proposed dcSR [6] that performs super-resolution on the I-frames of a video only, based on the insight that P- and B-frames can benefit from enhanced I-frames. They integrated dcSR with the H.264 decoder to exploit the benefit of super-resolution for I-frames only. dcSR requires that the decoding process be paused and wait for super-resolution to complete before resuming. FOCAS [25] uses a foveated, cascaded super-resolution method to produce the highest quality at the foveal region. However, it uses information about the user's eye gaze, which requires special hardware equipment to obtain.

Unlike prior works, in this paper, we propose to save computation by performing super-resolution on selected patches with high saliency in a video frame. We further design an optimization framework that can be used by an edge server for selecting patches for super-resolution when serving multiple users.

3 OVERVIEW OF EVASR

Figure 1 shows an overview of our proposed EVASR system that performs edge-based video delivery with salience-aware super-resolution. It has three main components: the video service provider that stores and streams videos to the users, the edge server that is capable of performing deep-learning-based super-resolution to enhance the quality of videos, and the clients who request and watch videos in high resolution.

In this work, we assume that the available downlink bandwidth from the video service provider is limited and thus may not support the transmission of high quality video to all users. On the other hand, the network bandwidth available between the edge server and the clients is very high and can even support streaming videos that are compressed in a lossless manner (e.g., via lossless H.264). This gives us the opportunity to exploit deep-learning-based super-resolution for upscaling the video and enhancing the video quality by leveraging computation resources available at the edge server.

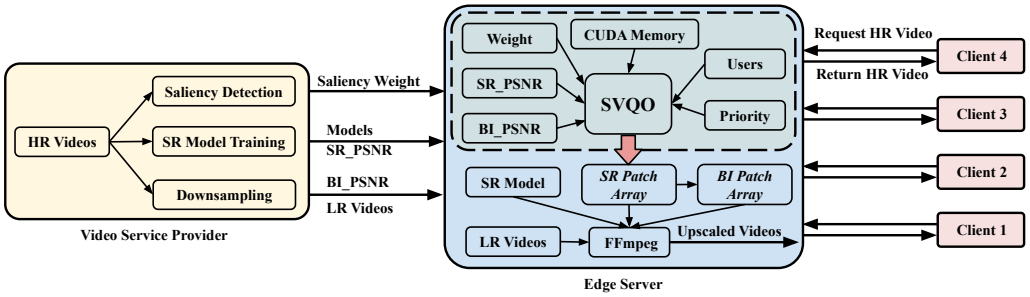


Fig. 1. Overview of our proposed EVASR system.

3.1 Edge Server

The edge server acts as a relay between the video service provider and the streaming client. It receives the low resolution video content from the video server. It then performs upscaling and video quality enhancements to increase the resolution and the visual quality of the received video. Finally, it sends the high resolution video to the client.

An edge server needs to serve multiple clients who may be watching different videos. However, given that only limited amount of computation resources are available at the edge server, it is challenging for the edge server to enhance the video quality for all users. To address this problem, we propose a patch-based scheme where deep-learning-based super-resolution is only performed for a selected set of patches in a video frame, while the remaining patches are upscaled using traditional, non-deep-learning-based approaches, such as bicubic interpolation.

The edge server has two main components: a saliency visual quality optimizer (SVQO) that decides the set of patches to perform super-resolution and a video frame processing tool that performs the upscaling (super-resolution or bicubic interpolation).

3.1.1 Saliency visual quality optimizer. The saliency visual quality optimizer (SVQO) formulates the patch selection problem as an optimization problem with the objective of maximizing the saliency visual quality for all users served by the edge server. We design the SVQO optimizer based on the following principles:

- Every edge server has its computational constraints, which means it can only serve a limited number of users at a time.
- Users with higher priority can obtain more computation resources and thus better video quality enhancements.
- Human eyes are more sensitive to regions on the video frame with higher saliency score. It is thus more important to improve the quality of the highly-salient areas on the video frame.
- Generally, upscaling via deep-learning-based super-resolution results in better video quality than upscaling via bicubic interpolation.
- However, applying super-resolution consumes more computational resources than applying bicubic interpolation.
- It is not necessary to apply deep-learning-based super-resolution to all patches in a frame.

Patches on a video frame have different saliency scores. With the recent success of deep learning, video salient object detection [26] can be used to extract the most visually distinctive objects in video sequences. Today, many existing models can produce high-quality spatial and temporal saliency information. Taking this into account, we design a new metric called the "saliency visual quality" that places more emphasis (i.e., weights) on the visual quality of areas on the frame where

human eyes are more sensitive to. The goal of our SVQO optimizer is to maximize the saliency visual quality subject to a number of constraints. To do so, it must also be aware of how much improvement can be obtained by performing super-resolution vs. bicubic interpolation (e.g., super-resolution PSNR (SR_PSNR) vs. bicubic PSNR (BI_PSNR)). Such information is provided as input to the optimizer to select the best set of patches for super-resolution. We describe details of the SVQO optimizer in Section 4.

3.1.2 Video Upscaling and Transmission. The SVQO optimizer solves for a binary array SR_Patch that encodes which patches are to apply super-resolution. We then need a dedicated process to perform the video upscaling operation.

The process decodes the received low resolution video, performs deep-learning-based super-resolution on selected patches and bicubic interpolation for the remaining patches. Finally, it transmits the upscaled high resolution video to the requested clients. The video scaling component is implemented as a new video filter in the FFmpeg [27]. We describe the implementation details in Section 5.

3.2 Video Service Provider

The video service provider stores high resolution videos (HR videos). It transcodes them into lower resolution videos (LR videos) for bandwidth-efficient downlink transmission and trains deep learning models for upscaling LR videos to HR videos. Since the SVQO optimizer at the edge server requires additional information about the video to formulate the optimization problem, in addition to the low resolution video, it also transmits video saliency information and visual quality results (e.g., SR_PSNR and BI_PSNR) to the edge server. The video saliency information can be computed offline at the video service provider, and the visual quality results can be obtained during deep-learning model training.

3.3 Clients

The edge server performs video upscaling to enhance the visual resolution and video quality. It then sends the high resolution video to the client, preferably in a lossless manner given that the available bandwidth between the edge and the client is very high.

Each client can have a different priority associated with them. A client with higher priority can potentially use more computational resources available at the edge server for super-resolution, thus receiving upscaled videos in better quality.

4 SALIENCY VISUAL QUALITY OPTIMIZER

In this section, we outline the design of the core component of EVASR, the saliency visual quality optimizer (SVQO). SVQO aims to find an optimal set of frame patches to perform deep-learning-based super-resolution for video quality enhancement, given the constraint of real-time super-resolution performance and the limited computation resources available. The objective is to maximize the saliency visual quality of video content displayed to the users.

Next, we first formulate our saliency visual quality optimization problem. We then describe the saliency video quality metric and saliency weight used in the problem formulation. Finally, we discuss other parameters in our optimization problem. We describe the variables used in our optimization problem formulation in Table 1.

4.1 Problem Formulation

Our goal to find the optimal set of patches for performing super-resolution is encoded in a binary array *SR_Patch*. In this array, 1s represent patches that apply deep-learning-based super-resolution

Table 1. Variables used in problem formulation.

C	total number of users that request at the same time
P_i	priority of user i
$SVQ_{i,j}$	saliency visual quality for user i watching video j
α	penalty parameter, details are described in Section 4.2.2
TP	total number of patches that will apply super-resolution among all users
TPF	total number of patches that will apply super-resolution in a frame
K_i	CUDA memory cost for user i
MK	maximum CUDA memory available at the edge server
MC	maximum number of users that can be supported at the same time while meeting real-time requirements
MP	maximum number of patches overall that can apply super-resolution at the same time to maintain real-time super-resolution
MPF	maximum number of patches that can apply super-resolution in a frame
F_j	total number of frames in video j
S_PSNR_k	saliency PSNR of frame k
W_p	saliency weight for patch p in a frame
SR_Patch	a binary array where 1s indicate patches to apply super-resolution and 0s indicate patches to apply bicubic interpolation for upscaling

for upscaling, and 0s in the array indicate patches that apply bicubic interpolation for upscaling. SR_Patch includes results for all users that are served by the edge server and all the videos these users are watching.

Given that upscaling via deep-learning-based super-resolution and bicubic interpolation will lead to different visual quality results, we can use the array SR_Patch to calculate the visual quality of the full upscaled frame (with some patches upscaled via super-resolution and others via bicubic). Considering that human eyes are more sensitive to the highly salient areas of a frame (e.g., areas with large amounts of motion), we assign different salient-based weights, W_p (Equation 5) to different patches and obtain a salient visual quality metric, SVQ (Equation 2). Equation 1 describes our objective to maximize the salient visual quality under resource and timing constraints:

$$\begin{aligned} \text{maximize:} \quad & \frac{1}{C}\sum_{i=1}^{C} P_i \times SVQ_{i,j} - \alpha(1-\frac{1}{C})TP \\ \text{subject to:} \quad & \sum_{i=1}^{C} K_i \le MK \\ & C \le MC \\ & TP \le MP \\ & TPF \le MPF \end{aligned} \tag{1}$$

In this formulation, i denotes the index of a user, and j denotes the index of video j that the user is watching. α represents a penalty term that relates to the number of users C, and we will discuss α in more detail in Section 4.2.2. The number of users C should not exceed MC, a constraint due to CUDA memory limit and the real-time inference requirement. MP is the maximum number of patches that can apply super-resolution at the same time while maintaining real-time super-resolution for

all users at the edge server, subject to the computation resources available at the edge server. We can obtain values of these constraints via a data-driven method.

4.1.1 Saliency Visual Quality. $SVQ_{i,j}$ in Equation 1 represents the saliency visual quality of user i while watching video j. We define SVQ_{ij} as the average saliency PSNR of all frames in video j:

$$SVQ_{i,j} = \frac{1}{F_j} \sum_{k=1}^{F_j} S\_PSNR_k \tag{2}$$

F_j is the total number of frames in video j, and S_PSNR_k stands for the saliency PSNR (defined in Equation 3) for frame k in video j. We represent the saliency PSNR of a frame as:

$$S\_PSNR = \sum_{p=1}^{\text{patch\_num}} W_p \times \left(SR\_Patch_p \times SR\_PSNR_p + (1 - SR\_Patch_p) \times BI\_PSNR_p \right) \tag{3}$$
$$SR\_Patch_p \in \{0, 1\}$$

Here, patch_num is the total number of tiled patches in a frame. For example, if we tile a frame into 6x6 patches, then the number of 36. W_p stands for saliency weight for patch p in the frame, computed offline by the video service provider. The saliency weight is obtained based on the saliency score. SR_Patch is a binary array we aim to solve for the optimization problem. The length of array SR_Patch is the same as the number of patches in the frame. SR_PSNR_p is the PSNR result for patch p obtained by applying super-resolution, and BI_PSNR_p is the PSNR result for patch p obtained by applying bicubic interpolation for upscaling.

4.1.2 Saliency Weight. The saliency weight W_p of patch p is computed based on the saliency score obtained from saliency detection (e.g., [28]). We first normalize the saliency score for each patch. Here, we use min-max normalization:

$$x'_p = \frac{x_p - min}{max - min}, p \in [1, \text{patch\_num}] \tag{4}$$

x_p denotes the original saliency score for patch p, computed by summing up all the saliency scores for each pixel in patch p. Based on the normalized saliency score x'_p, we apply the softmax function to convert the normalized saliency score for each patch into a probability distribution to represent the weight for each patch. We assign higher weights to patches with high saliency scores since human eyes are more attracted to contents in these areas.

$$W_p = \frac{e^{x'_p}}{\sum_{p=1}^{\text{patch\_num}} e^{x'_p}}, p \in [1, \text{patch\_num}] \tag{5}$$
$$\sum_{p=1}^{\text{patch\_num}} W_p = 1$$

4.2 Other Details

4.2.1 User Priority. In our problem formulation in Equation 1, we assign a different priority P_i to each user. When multiple users request video quality enhancement at the same time, an user with a higher priority can get more resources at the edge server and obtain better video quality improvements, e.g., with more patches upscaled via super-resolution as opposed to bicubic interpolation.

In our optimization problem, we consider the general scenario where different users request to stream different videos. If more than one users, e.g., $U = \{u_{i_1}, u_{i_2}, ..., u_{i_N}\}$, request to watch a same

video, then the enhanced video can be used by all n users. For example, each of the N users has a corresponding priority P_{i_j}, $j \in 1, 2, ..., N$. When these multiple users request a same video at the same time, we can simply consider them as one request by the user with highest priority. That is, $P_{i_{max}} = \max\{P_{i_1}, P_{i_2}, ..., P_{i_N}\}$. To facilitate delivery of enhanced frames to multiple users, the edge may need to temporarily cache these enhanced frames. However, since these users are requesting the same video at the same time, the enhanced frames do not need to be cached for too long.

4.2.2 Penalty Term and α. In our optimization objective shown in Equation 1, $\alpha(1 - \frac{1}{C})$ is a penalty term for the total number of patches that apply super-resolution, TP. The effect of this term is related to the number of users C. When there is only 1 user, $\alpha(1 - \frac{1}{C})$ becomes 0, and there is no penalty for increasing the number of patches to apply super-resolution. The edge server can apply super-resolution on as many patches as possible to achieve the best saliency video quality while subject to the constraints. However, as the number of users C increases, and since super-resolution requires more computational resources than bicubic, the coefficient term for penalty $(1 - 1/C)$ will increase to force the edge server to more carefully select patches to apply super-resolution. For example, consider a patch that can get 0.1 increase in saliency video quality by applying super-resolution. When C is small, the edge server may decide to apply super-resolution for this patch. However, when C gets larger, the edge side may be inclined to apply bicubic due to the larger penalty term for adding a patch to apply super-resolution.

The α parameter is used to balance the performance and computational resources. Larger α means that the edge side prefers to consider the resource constraint when assigning which patch to apply super-resolution. In this case, the increased total number of super-resolution patches (i.e., TP) would be more sensitive to the objective. Meanwhile, smaller α tells the edge side to emphasize on the improvement of the saliency video quality. In this case, the edge side may prefer to add more patches to improve the performance. For example, for a small α, even with 0.1 saliency visual quality improvement, the optimizer may choose to apply super-resolution. However, for larger α, the optimizer may choose to apply bicubic interpolation to save computational resources when the improvement is only 0.1. With larger improvements, e.g., 0.5, the optimizer would still assign this patch to apply super-resolution instead of bicubic. Thus, for edge devices that have less computation resources, we can set α to a larger value compared to when the edge devices are more powerful.

5 IMPLEMENTATION AND DEPLOYMENT

We implement EVASR based on the FFmpeg framework [27] and deploy it for use with not only desktop/laptop computers but also mobile devices such as mobile phones and single board computers (SBCs). To compare the per-frame processing time (i.e., time needed for upscaling) and the potential energy savings that can be achieved on mobile devices, we also implemented the baseline solutions where super-resolution is performed directly on the devices without involving the edge servers. We outline the implementation details of EVASR and baseline solutions in the sections below.

5.1 EVASR

We use the FFmpeg framework for performing video upscaling at the edge server. For each video that is being watched by a user, we create one FFmpeg process to be responsible for performing deep-learning-based super-resolution for patches selected in the *SR_Patch* array and bicubic interpolation for the remaining patches. Note that since different super-resolution models may be trained for each video, we are unable to use one FFmpeg process for all videos. To perform per-frame super-resolution, we adapt the FFmpegSR framework [9] we proposed in a prior work. FFmpegSR is implemented as an FFmpeg filter. It obtains low-resolution video frames from the video decoder in

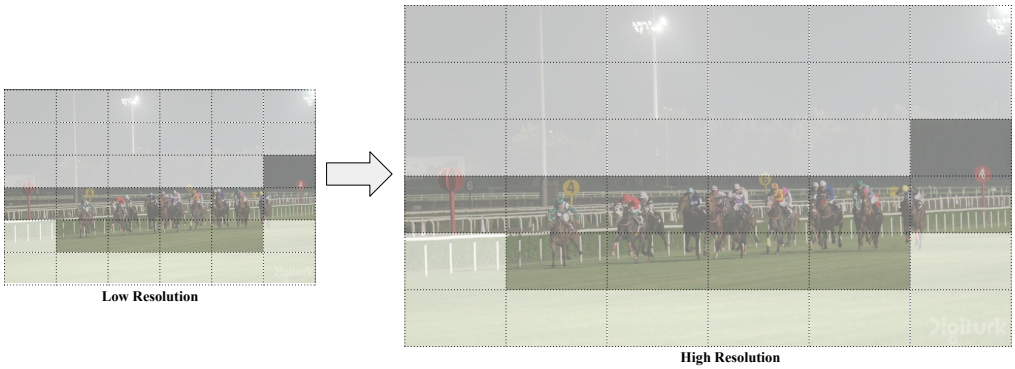


Fig. 2. This figure shows an example frame with 11 out of 36 patches (highlighted in the figure) performing super-resolution, while the remaining 25 patches are upscaled using bicubic interpolation. This example frame is from the "RaceNight" video in the UVG dataset [31].

Y/U/V channels as input, processes/upscales them, and forwards the output frame into the next step in the FFmpeg pipeline. We adapt FFmpegSR to upscale patches in each video frame using deep-learning-based super-resolution or bicubic interpolation based on the decision made by the SVQO optimizer.

As each frame is upscaled by the FFmpeg video filter, the high-resolution frames are streamed from the edge server to the client devices via the RTMP protocol [29]. In this way, we ensure that EVASR is compatible with a diverse set of client devices as long as they support streaming with the RTMP protocol.

FFmpegSR integrates Pytorch with FFmpeg using the PyTorch C++ API – LibTorch [30]. In this work, we use the ESPCN model for super-resolution in EVASR. We choose this model for its small model size, which is critical for real-time super-resolution, and its good visual quality performance. The trained ESPCN model is traced to the serialized torchscript model first. We then use CMake and LibTorch to generate a shared library `.so` file that can load and execute the torchscript model.

Unlike many super-resolution solutions that work in the R/G/B domain, FFmpegSR performs super-resolution/upscaling in the Y/U/V domain directly. This saves the extra computation needed for color format transformation. Given a frame, we perform bicubic interpolation using the `libswscale` library, available in FFmpeg. Patches that are selected to perform super-resolution are further sent to the model in a batch. That is, if n patches are selected in a frame, the batch size of super-resolution inference is n. Figure 2 illustrates the operation of the video filter. In this figure, out of 36 patches, only 11 patches (highlighted in the figure) are selected by SVQO to perform deep-learning-based super-resolution. In the FFmpeg video filter, we send these 11 patches to the super-resolution model as one batch with the batch size of 11. For patches that are upscaled by the super-resolution model, we use them to replace corresponding patches in the bicubic-interpolated full frame. In this way, regardless of how patches are upscaled, all patches have the same width and height and are composited onto a high resolution frame output by the video filter.

We calculate the saliency weight of each patch in each video based on saliency score obtained via a state-of-art real-time video saliency detection model, STVS [28]. To setup the optimization problem, we also need information about the PSNR results of patch upscaled via bicubic interpolation and deep-learning-based super-resolution, respectively (i.e., *BI_PSNR* and *SR_PSNR* in Equation 3). To do so, we use the FFmpeg filter we implemented to generate upsampled videos in two ways: via full-frame super-resolution and via full-frame bicubic interpolation, respectively. For each video, we tile each frame into 6x6 patches, 36 patches in total. For each patch (obtained via bicubic

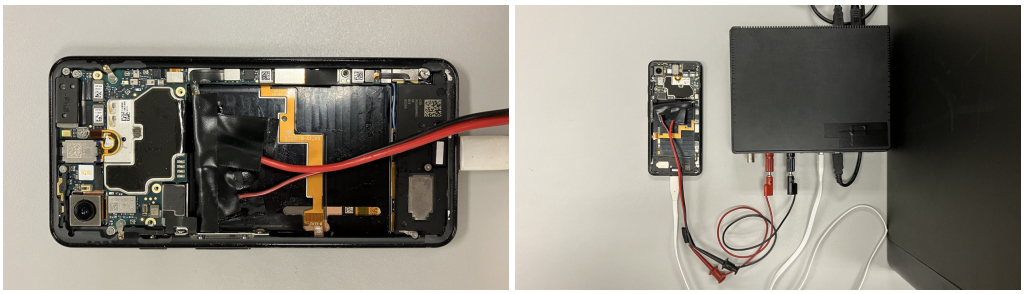


Fig. 3. Power Measurement Setup for Google Pixel 3. Power to the phone is directly supplied by the Monsoon Power Monitor.

interpolation or super-resolution), we calculate its PSNR value compared to the ground-truth high resolution video. The dimensions of BI_PSNR and SR_PSNR are exactly the same as our saliency weight, which is $Frame\_num \times Patch\_num$.

To solve the formulated optimization problem, we use the Gurobi Optimizer [32]. We define the decision variables, the linear objective function, the linear constraints, and use the Linear Programming solver in Gurobi to obtain results of the decision variable, i.e., the binary array SR_Patch_p. Given that an edge server can only perform super-resolution for a limited number of users, the scale of the problem is relatively small, which makes it possible for the optimizer to solve the problem within a very small amount of time. For example, our empirical results (as reported by Gurobi) show that the aggregated running time of Gurobi is 40 ms when solving the optimization problem for 12 seconds of videos and 290 ms when solving the problem for 5 minutes of videos.

5.2 Mobile-EVASR and MobileSR

We deploy EVASR with a Google Pixel 3 mobile phone [33] as the client device. We refer to this as the `Mobile-EVASR` implementation. In this setup, super-resolution is performed by EVASR Edge Server. We build an Android app that receives the upscaled video frames from the edge server via RTMP and directly displays the decoded raw frames (in upscaled, high resolution).

For comparison, we also implement a baseline `MobileSR` approach where the computation-intensive super-resolution operation is performed on the mobile device directly.

`MobileSR` utilizes FFmpeg for multimedia tasks and PyTorch for neural network (NN) tasks. Specifically, we developed a simple streaming video player application in Android with the C-based FFmpeg library. The C-based implementation in the application acts as the base layer to receive and decode the RTMP stream (in low resolution) from the server for video content. The C-based implementation also incorporates a callback function, which sends the decoded raw frame (e.g., raw frame in low resolution) back to the Java-based implementation in the application via the Java Native Interface (JNI) mechanism. The raw frame is then dispatched to the neural network model, managed by PyTorch for super-resolution. Subsequently, the upscaled frame is projected onto a GL-based surface within the Android User Interface (UI) for display.

We focus our comparison on the per-frame processing time (i.e., if real-time video playback can be achieved) and the power consumption on the mobile devices when using `Mobile-EVASR` vs. `MobileSR`. To measure the power consumption of the mobile phone during video streaming and playback, we disassemble the phone, tape up the battery (so that the power is not supplied by the battery), and use a Monsoon Power Monitor [34] for directly supplying power to the mobile phone. Figure 3 shows the power measurement setup for the Pixel 3 phone. We note that both `Mobile-EVASR` and `MobileSR` incur RTMP-based video streaming traffic on the mobile phone.

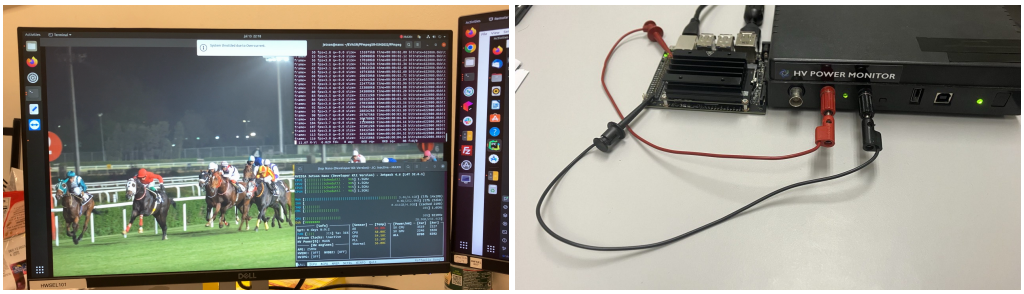


Fig. 4. Left: Video playback on Jetson via JetsonSR. A system notification can be seen on top of the screen that "System throttled due to Over-current". Right: Power Measurement Setup for Nvidia Jetson Nano.

5.3 Jetson-EVASR and JetsonSR

In addition to mobile phones, we also deploy EVASR on single board computer, Nvidia Jetson Nano Developer Kit [35], i.e., Jetson-EVASR. Jetson Nano has 128-core Maxwell GPU and Quad-core ARM A57 GPU with 4 GB memory. Here, we consider a setting similar to Google Chromecast [36], where a dongle is plugged into a screen/TV for video streaming and playback. In the Jetson-EVASR setting, super-resolution is performed by the EVASR Edge Server, and Jetson Nano receives the upscaled video frames via RTMP.

For comparison, we also implement a JetsonSR scheme where we install Ubuntu 20.04 on Jetson Nano and use FFmpegSR [9] for performing super-resolution on device directly. Figure 4 (Left) shows our JetsonSR setup for video super-resolution. Note in this figure that during super-resolution playback, a notification appeared in the top of the screen, indicating that "System throttled due to Over-current". This indicates very high power consumption.

To accurately measure the power consumption of Jetson Nano, we used the Monsoon Power Monitor to supply power directly. We notice that using the USB cable for power supply can cause voltage drop and thus affect the power consumption measurement. To reduce the impact of USB cable, we use the GPIO pin for power supply instead. Figure 4 (Right) shows the power measurement setup for Jetson Nano.

6 EVALUATION

6.1 Dataset

For evaluation, we used videos from the *UVG dataset* [31]. This dataset includes 16 test video sequences of various characteristics in 4K (3840x2160) resolution. These videos are captured at 50 and 120 fps. The length of these videos are 5 seconds and 12 seconds long. In our experiments, we used videos that are 50 fps and 12 seconds long. In addition, we reduced the frame rate of these videos from 50 fps to 25 fps as a frame rate between 24 fps and 30 fps are more common for streaming videos [37]. Among the 7 videos in the dataset that are 12 seconds long, we selected 4 most representative videos in scenes with different amount of motions to evaluate our proposed work: *RaceNight, FlowerKids, RiverBank, Twilight*. In addition, we also selected 4 videos from YouTube with longer duration. We select four 4K videos among four popular categories: *Haul, How-to, Vlog, and Challenge*. We refer to this as the *YouTube dataset*. We use the first 5 minutes of these 4 videos for evaluation. To demonstrate the generalizability of super resolution model, the model we used in the *YouTube dataset* is a publicly available super-resolution model without any fine-tuning.

We consider two different scaling factors in our study: x2 and x4. We generate low resolution videos (LR videos) by downsampling the videos in the dataset using bicubic downsampling to the low resolution of 960x540 (for x2 scaling) and 480x270 (for x4 scaling). We also create the high

Table 2. Comparison of upscaling performances among TensorFlow-based SR provided by FFmpeg (tf_SR), bicubic interpolation, and EVASR (ours). PSNR and S_PSNR results are in dB. Note that the results in this table are for full frame upscaling only.

video	upscaling	speed	PSNR	S_PSNR
RaceNight	tf_SR [38]	3.30x	43.05	39.94
	Bicubic	11.4x	43.08	39.62
	EVASR	3.76x	**44.33**	**42.16**
FlowerKids	tf_SR [38]	3.28x	**46.52**	43.60
	Bicubic	12.3x	46.17	43.38
	EVASR	3.90x	45.19	**44.83**
RiverBank	tf_SR [38]	3.31x	38.04	36.14
	Bicubic	11.7x	37.93	36.17
	EVASR	3.73x	**39.35**	**36.94**
Twilight	tf_SR [38]	3.30x	44.28	44.73
	Bicubic	14.1x	43.97	44.40
	EVASR	3.87x	**45.49**	**45.53**

resolution "groundtruth" video in 1920x1080 resolution and use it as the reference for PSNR and visual quality comparisons.

6.2 Comparisons Among TensorFlow-Based SR, Bicubic, and EVASR

The FFmpeg repository includes a naive implementation of a super-resolution video filter that uses TensorFlow (tf_SR) [38]. It supports two models: SRCNN [12] and ESPCN [17]. However, tf_SR can only perform full-frame super-resolution as it cannot support patch-based super-resolution like ours.

We compare our proposed system EVASR with tf_SR and bicubic interpolation. Since tf_SR can only perform full-frame super-resolution, for fair comparison, we only compare full-frame upscaling results. The upscale factor is x2. The input low resolution video is 960x540, and the output high resolution video is 1920x1080. Both tf_SR and our EVASR use the ESPCN [17] model for super-resolution. That is, the structure and number of parameters of the models are the same. For tf_SR, we directly used the TensorFlow ESPCN model used by FFmpeg. To show the potential of model fine-tuning, we recorded the parameters in the TensorFlow ESPCN model and used it to initialize the PyTorch model for EVASR. We took a further step to train the PyTorch model using each video as training data. This allows us to obtain better PSNR results with the PyTorch model used in EVASR. Note here that the purpose of this comparison is to show the potential of model fine-tuning and the visual quality improvements (e.g., PSNR) that can be achieved over bicubic interpolation. The visual quality results of EVASR should not be compared with tf_SR.

The results are shown in Table 2. In this table, the column "speed" represents the overall upscaling speed recorded by the FFmpeg pipeline against the video playback speed (e.g., 25 fps). The larger the value, the faster the speed is. A speed faster than 1x indicates real-time performance. The "speed" results throughout this paper are obtained on a machine using a GTX 3080 Ti GPU. "PSNR" represents the average of raw PSNR results obtained of each patch of each frame, and "S_PSNR" represents the saliency PSNR results.

We can see that the speed of EVASR is faster compared to tf_SR. Bicubic is the fastest upscaling method among the three. However, its visual quality performance is poor. Its PSNR and S_PSNR results always under-perform the deep-learning-based super-resolution methods. EVASR performs the best for all videos in both PSNR and S_PSNR metrics, with the exception of the PSNR metric of

Table 3. Impact of upscale factor. Note that the CUDA memory and speed results in this table are obtained using full-frame (i.e., all patches) super-resolution.

scale	parameters	size	CUDA memory	speed
x2	21,284	102KB	2491MB	3.76x
x4	24,752	115KB	2275MB	5.57x

Table 4. Visual quality results of full-frame x2 and x4 super-resolution via EVASR. PSNR and S_PSNR results are in dB.

Video	x2_PSNR	x2_S_PSNR	x4_PSNR	x4_S_PSNR
RaceNight	44.33	42.16	40.12	35.90
FlowerKids	45.19	44.83	39.92	39.92
RiverBank	39.35	36.94	36.01	33.83
Twilight	45.49	45.53	41.39	41.38

the "Flowerkids" video. After inspecting the video, we notice that there is an area in the video that has very high luminance, close to white. Our model cannot perform very well compared to the other two in this area. Nonetheless, our fine-tuned model outperforms others in saliency PSNR among all 4 videos.

6.3 Impact of Upscaling Factor

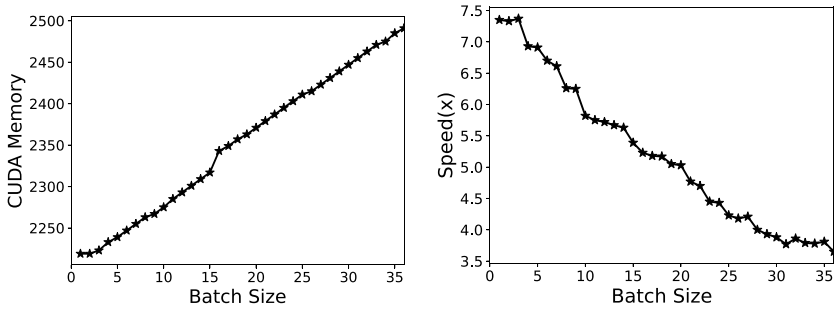
We considered two upscaling setups: x2 upscaling from 960x540 to 1920x1080 and x4 upscaling from 480x270 to 1920x1080. Table 3 compares the model size, CUDA memory, and super-resolution speed when using the ESPCN model with these two different scaling factors. The x2 model has fewer parameters and thus has slightly smaller model size. Since the x4 model works with 480x270 input videos, its CUDA memory usage is smaller compared to the x2 model. Its speed is also faster than the x2 model.

The smaller input video resolution of x4 upscaling, however, has a significant impact on the visual quality. Taking the video "RaceNight" as an example, we compare the PSNR and S_PSNR results in Table 4. We find that x4 upscaling results in 4 dB lower PSNR compared to x2 upscaling. We also note here that even though the visual quality performance of x4 super-resolution is not as good as x2, it is still better compared to x4 bicubic interpolation.

This analysis indicates that it is possible to use x4 scaling to further save the bandwidth (e.g., downloading video in the lower 480x270 resolution) and the computation resource (e.g., less CUDA memory and faster super-resolution inference speed) for users who might have lower priority.

6.4 Impact of Batch Size

We designed experiments to find the relationship between batch size and the inference time as well as CUDA memory consumption. The results are obtained on the "RaceNight" video with the batch size ranging from 1 to 36. Figure 5(a) shows the relationship between CUDA memory consumption and the batch size. It is not surprising that as the batch size increases (i.e., perform super-resolution for more patches in the frame), the CUDA memory consumption increases. Figure 5(b) shows that the batch size can substantially impact the upscaling speed in the FFmpeg pipeline. Note that here we are showing the speed when the edge server is serving only one client. As the batch size increases from 1 to 36, the speed decreases by half from 7.35x to 3.65x.



(a) Batch size vs. CUDA memory (b) Batch size vs. Speed

Fig. 5. Impact of batch size on CUDA memory and speed

Table 5. Impact of parameter α on the visual quality performance and the total number of patches to apply super-resolution.

Value of α	1e-1	1e-2	1e-3	1e-4	1e-5	1e-6	1e-7	1e-8	1e-9	1e-10
SVQ	40.8929	40.8929	41.0191	42.2007	42.3848	42.4014	42.4067	42.4068	42.4068	42.4068
Objective (Equation 1)	40.8929	40.8929	40.9299	41.7536	42.2986	42.3876	42.4042	42.4066	42.4068	42.4068
Total # of SR patches (TP)	0	0	119	5,962	11,489	18,398	33,865	36,265	37,606	37,895

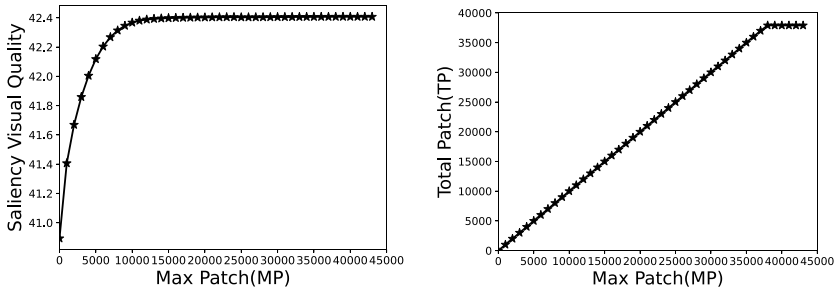
6.5 Analysis of the Optimizer

In this subsection, we report results from experiments to demonstrate the effectiveness of the saliency video quality optimizer (SVQO). Here, we consider the scenario where $C = 4$ users are requesting 4 videos of different characteristics in our experiment. The frame number for each video is $F = 300$. In our scenario, the maximum number of patches that our device can support per frame MPF with C is 36. Since we use 6x6 patches, performing super-resolution for 36 patches is the same as full-frame super-resolution. The maximum number of patches that can apply super-resolution is $C \times F \times MPF = 43,200$. In an ideal situation where there is no computation or real-time constraints, we can perform super-resolution on the maximum number of patches.

6.5.1 Impact of Parameter α. We conduct experiments to show the impact of parameter α. Here, the max_patch MP is set to be 43,200, and the number of users C is 4 with 4 distinct video contents. We set α from 1e-1 to 1e-10 and compare the average saliency PSNR (S_PSNR), saliency video quality (SVQ), and the objective of our optimization problem formulated in Equation 1.

The results are shown in Table 5. When α is large, the penalty may be too large, causing no patches to be assigned to apply super-resolution. As α decreases, the total number of super-resolution patches increases. As a result, the saliency visual quality improves. Specifically, we observe that when α decreases from 1e-4 to 1e-5, the saliency visual quality greatly improves with a moderate increase of total patches TP. However, when α decreases from 1e-6 to 1e-7, even with a very large increase of TP, the improvement of saliency visual quality is very limited, e.g., 0.0055. As α further decreases, even a very tiny increase in visual quality performance requires a significant amount of computational resources, which can be ineffective. Thus, we conclude that α =1e-5 achieves the best efficiency with performance and computational resources. We also consider α =1e-6 to compare the different levels of trade-off between performance and computational resource usage.

6.5.2 Impact of Max Number of Patches (MP). We know from the previous discussion that α has a significant impact on the total number of patches to apply super-resolution, TP, which affects the performance. To eliminate the impact of α as we investigate the impact of the MP parameter, we



(a) Max Number of Patches (*MP*) vs. SVQ (b) Max Number of Patches (*MP*) vs. Total Number of Patches (*TP*)

Fig. 6. Impact of *MP* on SVQ and *TP*

Table 6. Impact of user priority on saliency visual quality (SVQ).

Priority	RaceNight	FlowerKids	RiverBank	Twilight	Avg.
[1, 1, 1, 1]	42.1499	44.9532	36.8971	45.5390	42.3848
[1, 1, 1, 3]	42.1475	44.9518	36.8920	45.5443	42.3839
[1, 1, 1, 5]	42.1453	44.9507	36.8882	45.5460	42.3826
[1, 1, 5, 5]	42.1238	44.9347	36.9109	45.5427	42.3780
[1, 5, 5, 5]	42.1149	44.9553	36.9060	45.5419	42.3795

set α to 1e-10. Since α is very small, nearly zero, the optimization objective in Equation 1 is almost the same as saliency video quality (SVQ). We thus only show the impact of *MP* on SVQ in Figure 6(a). When *MP* increases from 0 to around 10,000, the SVQ result improves substantially from 40.9 to over 42.3. However, the improvement is very small beyond 10,000. The impact of *MP* on *TP* is shown in Figure 6(b). While *TP* typically grows larger with *MP*, it stops increasing once reaching a threshold. This indicates that even with enough computational resources, there is no need to apply super-resolution to some patches. These patches can obtain good enough visual quality results via bicubic interpolation directly.

6.5.3 Impact of User Priority. To investigate the impact of user priority P_i, we set α =1e-5 and set the maximum number of patches *MP* to 11,489. We chose this value because 11,489 is the total number of patches *TP* calculated by the SVQO optimizer when α =1e-5 and all users are set to the same priority (Table 5).

We consider five combinations of priority levels: [1,1,1,1], [1,1,1,3], [1,1,1,5], [1,1,5,5], and [1,5,5,5]. Table 6 shows the corresponding saliency video quality for each video, and Table 7 shows the total number of patches in each video to apply super-resolution. Note that since we consider each user watching a different video, each priority value not only corresponds to a user priority, but also corresponds to the video that the user watches.

Table 6 and Table 7 show that with higher priority, a user can get more super-resolution patches and thus obtain higher saliency video quality. For example, with priority levels set to [1,1,1,3], user #4 has higher priority than the remaining users. Comparing the first two rows in Table 7, we can see that user #4 watching the video "Twilight" can use more resources at the edge server for performing super-resolution for 268 more patches and thus obtain better SVQ results. Moreover, users with lower priority can still obtain good saliency video quality, despite a very slight decrease due to fewer super-resolution patches.

Table 7. Impact of user priority on the total number of patches TP to apply super-resolution.

Priority	RaceNight	FlowerKids	RiverBank	Twilight	Total
[1, 1, 1, 1]	3,486	3,594	2,398	2,011	11,489
[1, 1, 1, 3]	3,414	3,553	2,243	2,279	11,489
[1, 1, 1, 5]	3,358	3,526	2,148	2,457	11,489
[1, 1, 5, 5]	3,013	3,278	3,025	2,173	11,489
[1, 5, 5, 5]	2,918	3,678	2,763	2,130	11,489

Table 8. Impact of user priority on VMAF [39].

Priority	RaceNight	FlowerKids	RiverBank	Twilight	Avg.
[1, 1, 1, 1]	99.91	97.88	78.94	88.15	91.22
[1, 1, 1, 3]	99.91	97.85	78.85	88.46	91.27
[1, 1, 1, 5]	99.91	97.83	78.81	88.81	91.34
[1, 1, 5, 5]	99.91	97.68	80.22	88.33	91.54
[1, 5, 5, 5]	99.91	97.94	79.49	88.27	91.40

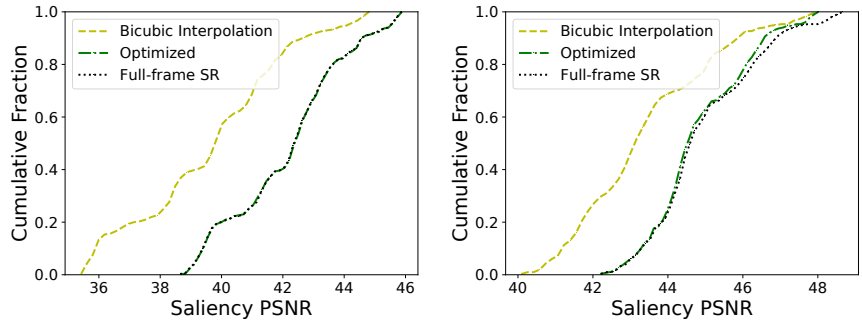
In addition to SVQ results, we also use another visual quality metric, video multi-method assessment fusion (VMAF [39]), an objective video quality metric developed by Netflix. We obtain the VMAF results as follows: we use EVASR for performing salience-aware super-resolution, that is, super-resolution for a set of patches selected by the SVQO optimizer and bicubic interpolation for the remaining patches. We save the output video in a lossless manner and compare it with the groundtruth high resolution video using the VMAF filter provided by FFmpeg. The results are shown in Table 8. We find that the VMAF results are consistent with the SVQ results.

6.6 Visual Quality Comparison

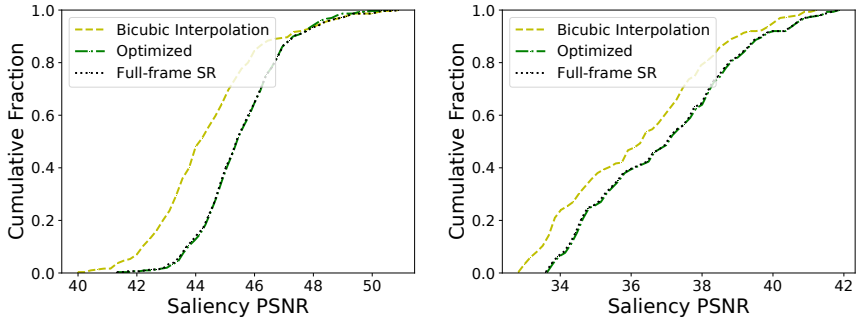
We evaluate EVASR using two different video sets: 4 videos in the UVG dataset, each 12 seconds long, and 4 videos in the YouTube dataset, each 5 minutes long. For the UVG dataset, we present results on two different α values: 1e-5 and 1e-6. For the YouTube dataset, we present results when α is set to 1e-6.

6.6.1 Per-frame Saliency PSNR. We compare the per-frame saliency PSNR results obtained from three methods: bicubic interpolation, full-frame deep-learning-based super-resolution, and the results from our EVASR using the saliency video quality optimizer, SVQO. In these experiments, all users are set to the same priority. We set α =1e-5, and the maximum number of patches MP is set to 43,200.

Figure 7 shows the distribution of per-frame saliency PSNR of all four videos in the UVG dataset. For each video, we compute the saliency PSNR for each frame. According to Table 5, when α =1e-5, a total of 11,489 out of 43,200 patches are selected to perform super-resolution. As we can see, with only $\approx 1/4$ patches applying super-resolution, our EVASR is able to reach the best visual quality performance. Across all four videos, our optimized results are significantly better than bicubic interpolation. For example, for "RaceNight", the median saliency PSNR value is 39.82 dB, 42.36 dB, and 42.38 dB for bicubic interpolation, our EVASR, and full-frame super-resolution results, respectively. Our results are very close to performing full-frame super-resolution. For "FlowerKids", our optimized results can even outperform the full-frame super-resolution results. This is because some areas in the video are overexposed, and several patches in the background are white. The bicubic PSNR value in these areas can reach over 100 dB. Our SVQO optimizer assigns these



(a) RaceNight (b) FlowerKids



(c) Twilight (d) RiverBank

Fig. 7. This figure shows the distribution of per-frame saliency PSNR of all four videos. In this figure, "Optimized" represents the results of our EVASR what uses the SVQO optimizer for selecting a set of patches for super-resolution.

patches to perform bicubic interpolation, which achieves maximum visual quality and also saves computational resources.

6.6.2 Saliency Video Quality Comparison. Based on saliency PSNR results, we can obtain saliency visual quality (SVQ) results for each video in the UVG dataset in Table 9. We can see that our optimized saliency video quality outperforms bicubic interpolation and even full-frame super-resolution on average saliency video quality. In this table, we further compare EVASR with two baseline approaches: Top-9 and Top-15. Top-9 represents performing super-resolution on a fixed number of 9 patches per frame with the highest saliency score. Given that when α =1e-5, EVASR performs super-resolution on 11,489 patches, this roughly translates to 9 patches per frame. When α =1e-6, EVASR performs super-resolution for 15 patches per frame on average, we thus compare EVASR (1e-6) with the Top-15 approach. Results in this table show that Top-9 and Top-15 do not perform as well as our EVASR approach. This shows that our SVQO optimizer is effective.

We also compute the saliency visual quality (SVQ) for the 4 longer (5 minutes) YouTube videos. The results are shown in Table 10. We note here that for these 4 videos in the YouTube dataset, we did not fine-tune the deep-learning-based super-resolution model via re-training. Instead, we directly used the parameters in the TensorFlow ESPCN model provided by FFmpeg tf_SR. The results show that even when using a general model, our SVQO optimizer can still achieve the best performance without large-scale pre-training.

Table 9. Saliency video quality (SVQ) comparison among 4 UVG videos.

SVQ	RaceNight	FlowerKids	RiverBank	Twilight	Avg
Bicubic	43.3757	36.1723	39.6214	44.4020	40.8929
Full-frame SR	44.8347	36.9358	42.1632	45.5302	42.3660
Top-9	43.9910	36.3116	40.1280	45.4974	41.4820
EVASR (1e-5)	44.9532	36.8971	42.1499	45.5390	42.3848
Top-15	44.2602	36.5261	40.3651	45.5181	41.6674
EVASR (1e-6)	44.9580	36.9316	42.1606	45.5553	42.4014

Table 10. Saliency video quality (SVQ) comparison among 4 YouTube videos.

SVQ	Haul	How-to	Vlog	Challenge	Avg
Bicubic	42.31	39.76	38.11	40.89	40.2675
Full-frame SR	44.33	39.74	38.09	40.61	40.6925
EVASR (1e-6)	49.95	42.23	41.23	46.58	44.9975

Table 11. VMAF results comparison among 4 UVG videos.

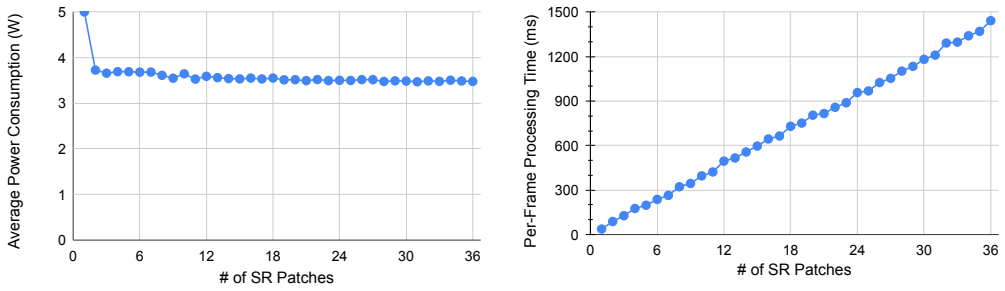
VMAF	RaceNight	FlowerKids	RiverBank	Twilight	Avg
Bicubic	98.9517	95.5286	84.9478	77.6817	89.2775
Full-frame SR	99.9857	99.8346	96.4589	91.5584	96.9594
Top-9	99.9297	96.9174	79.3394	88.2174	91.1010
EVASR (1e-5)	99.9106	97.8755	88.1485	78.9434	91.2195
Top-15	99.9741	98.1345	82.1510	89.8296	92.5223
EVASR (1e-6)	99.9843	98.2353	93.4733	86.1951	94.4720

Table 12. VMAF results comparison among 4 YouTube videos.

VMAF	Haul	How-to	Vlog	Challenge	Avg
Bicubic	86.27	81.15	79.77	83.26	82.6125
Full-frame SR	88.20	83.80	81.77	84.78	84.6375
EVASR (1e-6)	86.80	82.78	80.54	83.91	83.5075

6.6.3 VMAF Results. Table 11 compares the VMAF results of different upscaling approaches for all 4 UVG videos. We can see that full-frame super-resolution achieves the best VMAF results. For "Racenight" and "FlowerKids", the differences among different approaches are relatively small. However, for the remaining two videos, full-frame super-resolution is significantly better than bicubic interpolation. When α is set to 1e-6, our EVASR can also achieve great performance while saving computational resources, e.g., applying super-resolution for only $\approx 40\%$ of all patches. In addition, EVASR outperforms Top-9 and Top-15 approaches, achieving higher average VMAF results compared to them.

We also compared VMAF for all 4 YouTube videos in Table 12. Using the model provided by FFmpeg that is not trained on each specific video, the VMAF results of super-resolution still outperform bicubic results. Even though our optimization objective is not VMAF, we can still achieve large VMAF improvements with only small number of patches applying super-resolution. This indicates our optimizer is able to select most important patches for super-resolution, which is useful under computational constraints.



(a) As the number of super-resolution patches per-frame increases, the mobile phone quickly overheats and throttles its power consumption.

(b) The per-frame processing time increases linearly with the number of patches that perform super-resolution in a frame.

Fig. 8. Results of MobileSR on Pixel 3 mobile phone show that MobileSR can barely support real-time playback when performing super resolution for only 1 out of 36 patches in the video frames.

6.7 Per-Frame Processing Time and Power Savings on Mobile Devices

6.7.1 Results on Pixel 3 Mobile Phone. Figure 8 shows the results of using the baseline MobileSR for performing super-resolution on the Pixel 3 mobile phone directly. The video used in this experiment is *RaceNight* from the *UVG dataset*. The video is 12 seconds long at 25 frames-per-second with a total of 300 frames. We tile each video frame into 6x6 patches (i.e., 36 patches in total). In this figure, the x-axis represents the number of patches in a frame that we upscale using super-resolution. We report in this figure how average power consumption changes as the number of super-resolution patches increases. Figure 8(a) shows that when super-resolution is performed for only 1 out of 36 patches in a frame, the average power consumption of the Pixel 3 mobile phone is 5 W. However, as the number of super-resolution patches increase, the device quickly overheats, and the operating system throttles the power consumption to under 4 W to avoid damage to the device.

Figure 8(b) further shows the average per-frame processing time in milliseconds as the number of super-resolution patches increase. Ideally, the per-frame processing time should be small enough to support real-time playback. For example, given that our testing video is 25 frames-per-second, the processing time of each video frame should not be longer than 40 ms. Results show that when performing super-resolution on only 1 out of 36 patches, the per-frame processing time is 37 ms, which can barely support real-time playback. When the number of super-resolution patches increase, however, this processing time increases linearly. This is a combination of both the limited computation resource available on the Pixel 3 mobile phone and the thermal throttling due to overheating.

Figure 9 shows the total energy consumption of Mobile-EVASR when playing back a 12 seconds long video. With EVASR, computation-intensive super-resolution is performed on the edge server and streamed to Pixel 3 via the RTMP protocol. This results in significant power savings – the average power consumption is only 1.8 W. In addition, the energy consumption on the Pixel 3 phone is consistent regardless of the number of super-resolution patches. In contrast, with the increased processing time per-frame, the total energy consumption for playing the 12-second long test video is significantly higher and increases with the number of super-resolution patches. For example, consider the scenario where the edge resource is shared by 4 clients (i.e., the setting in Section 6.5), with EVASR (1e-6), the average number of super-resolution patches per-frame is about 15. In this setting, the total energy consumption of MobileSR is 632 J, compared to just 22 J with Mobile-EVASR.

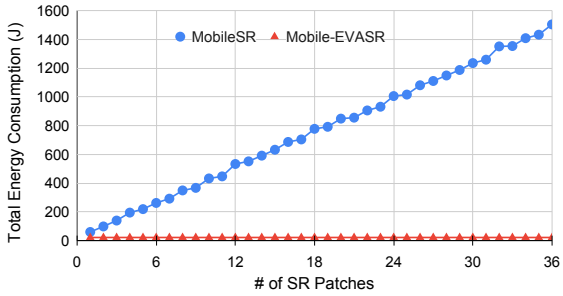
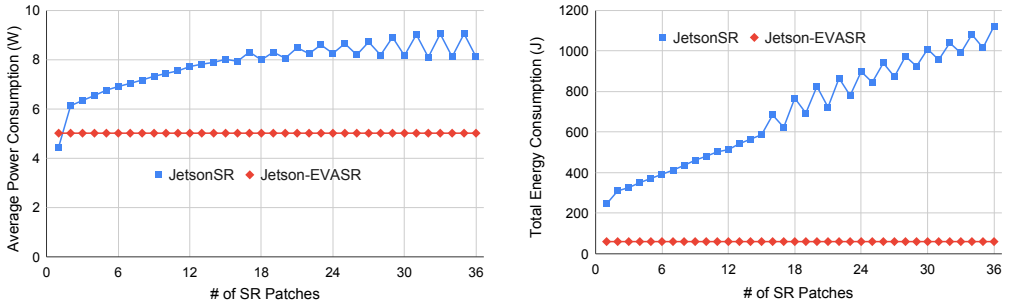


Fig. 9. Total energy consumption (in Joules) comparison between `MobileSR` and `Mobile-EVASR` on the Pixel 3 mobile phone when playing a 12-second long video with 300 frames in total.



(a) As the number of super-resolution patches per-frame increases, the average power consumption on Jetson Nano with `JetsonSR` increases sub-linearly due to thermal throttling; while with `Jetson-EVASR`, the average power consumption remains consistent around 5 W.

(b) With the increased per-frame processing time, the total energy consumption of playing back the video increases linearly with the number of super-resolution patches in `JetsonSR`; while with `Jetson-EVASR`, the total energy consumption is consistent about 60 J.

Fig. 10. Average power consumption (in Watts) and total energy consumption (in Joules) comparison between `JetsonSR` and `Jetson-EVASR` when playing a 12-second long video with 300 frames in total.

6.7.2 Results on Nvidia Jetson Nano. Figure 10(a) shows the average power consumption of Jetson Nano as the number of super-resolution patches increase from 1 to 36. When performing super-resolution on only 1 out of 36 patches, the average power consumption is 4.45 W. The power consumption quickly increases as more patches are upscaled via super-resolution on the Jetson Nano device. The system then quickly enters the throttling state (as shown in Figure 4 (Left)). As a result, the per-frame processing time increases with the number of super-resolution patches per-frame. As a result, the total energy consumption increases roughly linearly as shown in Figure 10(b). `Jetson-EVASR`, on the other hand, consistently consumes about 5 W power on Jetson Nano, and the total energy consumption for 12 seconds playback is only about 60 J, significantly smaller than the `JetsonSR` baseline.

6.8 Further Bandwidth Savings

It is possible to further reduce the bandwidth required between the edge and the client. Instead of streaming the full upscaled video to the client, we can choose to only stream the "diff" in pixel values in the Y channel only, namely the "mask". This assumes that the client is capable of performing the bicubic upscaling itself. It receives the "diffs" between bicubic and deep-learning-based super-resolution and applies the "diffs" to the bicubic-upscaled frame, thereby achieving the higher quality

Table 13. Size comparison: full video vs. mask-only

size (in MB)	RaceNight	FlowerKids	RiverBank	Twilight
Full video	261	240	263	192
Mask-only	150	148	136	90

Fig. 11. Visual comparison of six patches. These patches are extracted from video frames in the UVG dataset [31].

provided by super-resolution. To get a mask for each video, the edge computes the different pixel values between our optimized video frame and the video frame upscaled via bicubic interpolation. It then compresses the mask data as lossless .png files. Taking the 4 videos in the UVG dataset as an example, Table 13 shows that nearly 40% bandwidth savings can be achieved compared to directly streaming the full optimized video.

7 VISUAL RESULTS

Figure 11 shows the upscaling results of six patches from three different videos: "RiverBank", "RaceNight" and "FlowerKids". Each of these patches represents 1/36 of the video frame it was contained in. These patches are associated with relatively high saliency score. Thus, our EVASR selects them for super-resolution-based upscaling. In these images, bicubic interpolation results appear to be more blurry, while super-resolution patches have better visual quality and are visually closer to the groundtruth patch.

8 CONCLUSION

In this paper, we proposed EVASR, a system that performs edge-based video delivery to clients with salience-aware super-resolution. Unlike many prior works that perform full-frame super-resolution, we propose to efficiently use the computation resources available at the edge server by performing super-resolution on patches selected by a saliency visual quality optimizer (SVQO), while the remaining patches are upscaled via bicubic interpolation. We implemented our EVASR system and performed extensive evaluations on the visual quality, super-resolution speed, and energy savings that can be achieved via EVASR. Results show that the SVQO optimizer can effectively select a set of patches for super-resolution that improves both the saliency visual quality and the VMAF quality metric. EVASR can support real-time per-frame super-resolution to 1920x1080 and can support multiple clients' super-resolution requests simultaneously. In addition, EVASR can significantly reduce the energy consumption on mobile devices compared to baseline approaches.

ACKNOWLEDGMENTS

We appreciate constructive comments from anonymous referees. This work is partially supported by NSF under grants CNS-2200042 and CNS-2200048.

REFERENCES

[1] B. Bross, Y.-K. Wang, Y. Ye, S. Liu, J. Chen, G. J. Sullivan, and J.-R. Ohm, "Overview of the versatile video coding (vvc) standard and its applications," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 31, no. 10, pp. 3736–3764, 2021.

[2] Y. Chen, D. Murherjee, J. Han, A. Grange, Y. Xu, Z. Liu, S. Parker, C. Chen, H. Su, U. Joshi *et al.*, "An overview of core coding tools in the av1 video codec," in *2018 picture coding symposium (PCS)*. IEEE, 2018, pp. 41–45.

[3] H. Yeo, Y. Jung, J. Kim, J. Shin, and D. Han, "Neural adaptive content-aware internet video delivery," in *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, 2018, pp. 645–661.

[4] B. Lim, S. Son, H. Kim, S. Nah, and K. Mu Lee, "Enhanced deep residual networks for single image super-resolution," in *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, 2017, pp. 136–144.

[5] M. Dasari, A. Bhattacharya, S. Vargas, P. Sahu, A. Balasubramanian, and S. R. Das, "Streaming 360-degree videos using super-resolution," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2020, pp. 1977–1986.

[6] D. Baek, M. Dasari, S. R. Das, and J. Ryoo, "dcsr: practical video quality enhancement using data-centric super resolution," in *Proceedings of the 17th International Conference on emerging Networking EXperiments and Technologies*, 2021, pp. 336–343.

[7] H. Yeo, C. J. Chong, Y. Jung, J. Ye, and D. Han, "Nemo: enabling neural-enhanced video streaming on commodity mobile devices," in *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*, 2020, pp. 1–14.

[8] A. Borji and L. Itti, "State-of-the-art in visual attention modeling," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 1, pp. 185–207, 2012.

[9] N. Li and Y. Liu, "FFmpegSR: A General Framework Toward Real-Time 4K Super-Resolution," in *2022 IEEE International Symposium on Multimedia (ISM)*. IEEE, 2022, pp. 293–296.

[10] ——, "EVASR: Edge-Based Video Delivery with Salience-Aware Super-Resolution," in *Proceedings of the 14th Conference on ACM Multimedia Systems*, 2023, pp. 142–152.

[11] C. Dong, C. C. Loy, K. He, and X. Tang, "Learning a deep convolutional network for image super-resolution," in *European conference on computer vision*. Springer, 2014, pp. 184–199.

[12] ——, "Image super-resolution using deep convolutional networks," *IEEE transactions on pattern analysis and machine intelligence*, vol. 38, no. 2, pp. 295–307, 2015.

[13] C. Dong, C. C. Loy, and X. Tang, "Accelerating the super-resolution convolutional neural network," in *European conference on computer vision*. Springer, 2016, pp. 391–407.

[14] J. Kim, J. Kwon Lee, and K. Mu Lee, "Accurate image super-resolution using very deep convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 1646–1654.

[15] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[16] J. Kim, J. Kwon Lee, and K. Mu Lee, "Deeply-recursive convolutional network for image super-resolution," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 1637–1645.

[17] W. Shi, J. Caballero, F. Huszár, J. Totz, A. P. Aitken, R. Bishop, D. Rueckert, and Z. Wang, "Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 1874–1883.

[18] C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang *et al.*, "Photo-realistic single image super-resolution using a generative adversarial network," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4681–4690.

[19] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, 2014, pp. 2672–2680.

[20] Y. Zhang, K. Li, K. Li, L. Wang, B. Zhong, and Y. Fu, "Image super-resolution using very deep residual channel attention networks," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 286–301.

[21] Z. Lin, P. Garg, A. Banerjee, S. A. Magid, D. Sun, Y. Zhang, L. Van Gool, D. Wei, and H. Pfister, "Revisiting rcan: Improved training for image super-resolution," *arXiv preprint arXiv:2201.11279*, 2022.

[22] J. Liang, J. Cao, G. Sun, K. Zhang, L. Van Gool, and R. Timofte, "Swinir: Image restoration using swin transformer," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2021, pp. 1833–1844.

[23] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, "Swin transformer: Hierarchical vision transformer using shifted windows," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2021, pp. 10 012–10 022.
[24] J. Kim, Y. Jung, H. Yeo, J. Ye, and D. Han, "Neural-enhanced live streaming: Improving live video ingest via online learning," in *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, 2020, pp. 107–125.
[25] L. Wang, M. Hajiesmaili, and R. K. Sitaraman, "Focas: Practical video super resolution using foveated rendering," in *Proceedings of the 29th ACM International Conference on Multimedia*, 2021, pp. 5454–5462.
[26] H. J. Seo and P. Milanfar, "Static and space-time visual saliency detection by self-resemblance," *Journal of vision*, vol. 9, no. 12, pp. 15–15, 2009.
[27] "FFmpeg," http://www.ffmpeg.org/, 2023.
[28] C. Chen, G. Wang, C. Peng, Y. Fang, D. Zhang, and H. Qin, "Exploring rich and efficient spatial temporal interactions for real-time video salient object detection," *IEEE Transactions on Image Processing*, vol. 30, pp. 3995–4007, 2021.
[29] "Ffmpeg rtmp," https://ffmpeg.org/ffmpeg-protocols.html#rtmp.
[30] "INSTALLING C++ DISTRIBUTIONS OF PYTORCH," https://pytorch.org/cppdocs/installing.html, 2022.
[31] A. Mercat, M. Viitanen, and J. Vanne, "Uvg dataset: 50/120fps 4k sequences for video codec analysis and development," in *Proceedings of the 11th ACM Multimedia Systems Conference*, 2020, pp. 297–302.
[32] "Gurobi Optimization," https://www.gurobi.com/, 2023.
[33] "Pixel phone hardware tech specs," https://support.google.com/pixelphone/answer/7158570?hl=en.
[34] "Monsoon power monitor," https://www.msoon.com/high-voltage-power-monitor.
[35] "Nvidia jetson nano," https://developer.nvidia.com/embedded/jetson-nano-developer-kit.
[36] "Google chromecast," https://store.google.com/us/product/chromecast_google_tv?hl=en-US.
[37] "The importance of frame rate in moviemaking." https://www.adobe.com/creativecloud/video/discover/frame-rate.html, 2023.
[38] "FFmpeg sr Filter," https://ffmpeg.org/ffmpeg-filters.html#sr-1, 2023.
[39] "Toward A Practical Perceptual Video Quality Metric," https://medium.com/netflix-techblog/toward-a-practical-perceptual-video-quality-metric-653f208b9652, June 2016.