dcSR: Practical Video Quality Enhancement Using Data-Centric Super Resolution

Duin Baek, Mallesham Dasari, Samir R. Das, and Jihoon Ryoo†
Stony Brook University, †SUNY Korea
{dback,mdasari,samir,jiryoo}@cs.stonybrook.edu

ABSTRACT

With the next generation immersive video applications, network capacity is becoming a growing bottleneck to deliver a high quality video to end-users. Recent advances to tackle this challenge introduced super-resolution (SR) for video quality enhancement through neural computations by leveraging client-side compute capacity. However, the existing SR models are bulky, compute-, and memory-expensive, which makes it difficult to deploy them in practice. In this work, we present dcSR, a lightweight data-centric SR approach that enables a practical neural quality enhancement for videos. On the server-side, dcSR constructs micro SR models trained on a few selected frames from each video through a data-centric paradigm by employing a long term video scene understanding mechanism. On the client-side, dcSR integrates the micro SR models into the regular video decoder and enhances the video quality in real-time without compromising on quality enhancement. We evaluate dcSR and show its benefits by comparing it with previous methods.

CCS CONCEPTS

• **Information systems** → **Multimedia streaming**; • **Computing methodologies** → **Reconstruction**; • **Human-centered computing** → **Ubiquitous and mobile devices**;

KEYWORDS

Video Streaming, Video Compression, Super Resolution

ACM Reference Format:
Duin Baek, Mallesham Dasari, Samir R. Das, and Jihoon Ryoo†. 2021. dcSR: Practical Video Quality Enhancement Using Data-Centric Super Resolution. In *The 17th International Conference on emerging Networking EXperiments and Technologies (CoNEXT '21), December 7–10, 2021, Virtual Event, Germany.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3485983.3494856

1 INTRODUCTION

Internet video has seen a rapid growth in multitude of ways in recent years— users, applications, and network protocols. However, with the evolution of current generation streaming applications (e.g., Netflix or YouTube) and future generation immersive applications (e.g., AR/VR and 360° video), the demand for network capacity is increasing exponentially [3]. While the applications and the Internet protocols are constantly evolving in parallel, delivering the best video quality to end users remains a challenging problem.

Though various advances have improved the video quality of experience (QoE) [18, 23], majority of the work relies on a single resource dimension— *network capacity*, resulting in a fundamental limitation where users with poor network capacity are always delivered a poor quality video. The problem becomes more acute with the increased number of end-users competing for the shared network resource.

To overcome the above limitations, recent advances in this space introduced a neural video quality enhancement paradigm by leveraging an additional client-side resource dimension— *compute capacity* along with network capacity [7, 26, 27]. In particular, most of the related works use a deep learning based super-resolution (SR) approach [7, 26, 27], where a client with poor network capacity downloads a low quality video, which is then enhanced through neural computations before the playback.

Practical Challenges. Given the growing compute capacity on client devices (e.g., GPUs/NPUs) [6], neural enhancement appears to be a natural fit to improve the end-user experience. However, realizing the state-of-the-art SR-driven streaming methods in practice poses two nontrivial challenges: 1) current SR models are extremely expensive in terms of *computation and memory*, 2) these models suffer from *generalization-specialization trade-off*. The first challenge mainly arises from the need to train bulky SR models to achieve good quality enhancement. The second challenge is more general and attributed to either **1)** training the SR model on standard datasets with an aim to generalize across all videos, consequently improving little quality on unseen video content [27] or **2)** tailoring a SR model specific to each video at the cost of downloading the model along with the corresponding video on-demand [26, 27]. Thus, the existing SR methods cannot be easily adopted in practice for today's real-time on-demand video streaming on compute constrained client devices.

dcSR. We propose dcSR (Data-Centric SR), a practical super-resolution approach for video streaming that overcomes the above limitations and enables real-time neural video quality enhancement on heterogeneous client devices. dcSR, in its core relies on a data-centric AI paradigm [8], where SR models are trained with a special understanding of the video data. Unlike the existing model-centric approaches [26, 27] that optimize models to deal with the noise in the data, dcSR targets improving the data consistency for training. To this end, dcSR constructs several lightweight micro SR models for each video by automatically analysing the long term spatio-temporal redundancy in the video and applying SR on a few selected

[*] Jihoon Ryoo is the corresponding author.

frames. The resulting system is one that effectively minimizes the client-side computation (i.e., inference) time without compromising on the video quality enhancement.

Insights. The key insight behind dcSR is the frame structure followed by the current generation video codecs [24], called group of pictures (GOP) structure classifying each frame as I, P, or a B frame. During the encoding, to exploit the redundancies across the frames for compression efficiency, these frames refer to each other to encode *residual*, which is the difference between two pixel blocks instead of the entire pixel block. For this, while I frames do not make reference to any frame, P frames make reference to I or P frames. Similarly, B frames make reference to previous and future frames [24]. The insight here is that *enhancing the I frames alone is sufficient because the P and B frames are automatically enhanced as they depend on I frames.* Additionally, we also observe that videos with longer duration tend to have repeated GOP structures, meaning the scenes in the video repeat sometimes resulting in similar I frames but at longer time intervals (see §3.1). This suggests another insight that *the model trained for one I frame can be reused for other similar I frames.*

Using the above insights, on the server-side, dcSR constructs several lightweight *micro SR models* for each video with each model corresponding to a selected group of I frames (§3.1). We realize this by using a *feature extraction and clustering* mechanism to group visually similar I frames, and train each micro model for each cluster of I frames. The resulting system is one that optimizes the number of models for each video and the model parameters without compromising on the quality enhancement, which effectively resolves the inference and model download challenges. On the client-side, we develop a *video decoder integration pipeline* that seamlessly enhances I-frames in the decoded picture buffer and caches the models for future I frames that belong to the same cluster (§3.2).

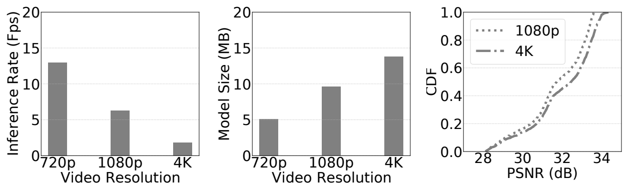
We have built dcSR prototype on top of FFMPEG [9], a multimedia framework with standard video codecs (e.g., H.264/265), and evaluated its performance across diverse contents, video resolutions, and devices. Our evaluation shows that dcSR can perform SR in real-time (>30FPS) across diverse devices without compromising on video quality enhancement for the same bandwidth, compared to state-of-the-art methods. We also show that dcSR reduces bandwidth usage by 25% for the same video quality compared to previous methods. In summary, we make the following contributions:

- We design and develop a practical super-resolution approach for real-time neural video quality enhancement.
- We build SR-FFMPEG by integrating dcSR into H.264 decoder implementation from FFMPEG. We are releasing SR-FFMPEG, hoping it brings value to the community in fostering research in this space [2].
- We comprehensively evaluate and showcase the benefits of dcSR by comparing with state-of-the-art SR methods.

2 BACKGROUND AND MOTIVATION

2.1 Client-side Video Quality Enhancement

High quality video applications are fundamentally limited by network capacity. When network capacity is limited, the clients have no other choice than to download and display poor quality videos. To overcome the above fundamental limitation of single dimensional resource (i.e., poor network capacity) in video content delivery, recent advances introduced a video quality enhancement paradigm using neural super-resolution models [26, 27] by leveraging client-side *compute capacity* as an additional resource. The key idea of super-resolution is to reconstruct a high-resolution (quality) video from its lower-resolution (quality) version. Recent developments in the computer vision literature has seen tremendous success in super-resolving low quality videos using convolutional neural networks (CNN) [1, 11]. However, deploying these SR models in practice for on-demand streaming brings several nontrivial challenges as discussed below.



(a) Inference Rate (b) Model Overhead (c) Quality Variance

Figure 1: Challenges of realizing SR in practice

2.2 Practical Challenges of Video SR

Ideally, a universal SR model that can enhance the quality of any video can eliminate the need for downloading a model every time a client requests a video. However, given that the Internet video ecosystem is vast and new contents are continuously generated, generalizing a single model for all videos that can provide adequate quality enhancement is likely infeasible. Therefore, we choose to construct SR models specialized for each video at the cost of downloading the corresponding model along with the video. Video specific SR models are more efficient because they can overfit on the extracted nonlinear relations among the pixels for a particular video and greatly enhance the video even from a relatively low quality. While video-specific SR model is indeed promising for efficient quality enhancement, realizing it in a practical setting poses several nontrivial challenges. To illustrate this, we run a series of benchmarks using the experimental setup described in §4. We train a DNN model similar to a recent SR model, NAS [27]. Figure 1 shows the impact of video resolution on model size and inference rate.

High inference latency: Figure 1(a) shows the impact of video resolution on inference speed. The SR inference is less than 15 FPS for all resolutions. However, for a good user experience, the player has to display the video at least 30 frames per second, which is not possible with the existing SR models. As we show in §4, this high inference also leads to excessive power consumption because of the frequent and heavy usage of GPU for neural computations.

Model download overhead: Figure1(b) shows that for a given quality, the model size grows significantly with the increase in resolution. Moreover, training a single SR model for the entire video leads to inefficient utilization of the network capacity as the model needs to be downloaded in the beginning of the streaming which leads to even when the users do not watch some part (e.g., ending credit) of the video.

Large quality variance: The third challenge is when trained on longer videos, SR models try to generalize across the entire video, making it to difficult to retain the details of each frame uniformly. As

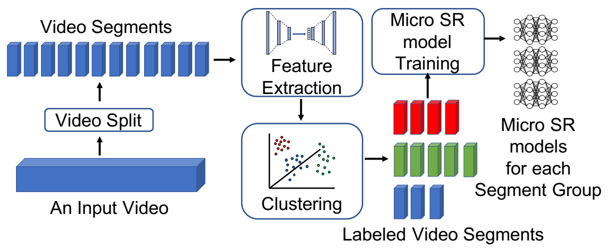


Figure 2: Workflow of Server-side dcSR

a result, for a given model, the quality enhancement varies by almost 5dB even when trained on a single video (≈ 12 mins). Such frequent fluctuations in quality can hurt the user experience [18, 23].

3 SYSTEM DESIGN

dcSR consists of two components. First, on the server-side, dcSR constructs a few micro SR models for each video based on the insights gathered from video codecs and its content. Note that a micro SR model has a simpler model architecture trained on only a few video frames, which makes the overall number of trainable hyperparameters less than those in the literature [26, 27]. Second, on the client-side, dcSR seamlessly reflects quality of SR-enhanced frames to the rest of the frames.

3.1 Server-side dcSR

Figure 2 shows the pipeline of our server-side dcSR design that includes: 1) video split and feature extraction, 2) video segment clustering, and 3) training micro models. We explain each of these components in the following.

3.1.1 Video split & feature extraction. On the server-side, raw videos are encoded using standard video codecs (e.g., H.264 [12]) to reduce the bandwidth. Each frame is classified into either I, P, or a B frame. The P and B frames are mostly reconstructed from I frames. Thus, they consume much lower bitrate, while I frames have a higher bitrate. Most of the existing works [26, 27] split a video into multiple fixed length segments to reduce the complexity for adaptive bit rate algorithms. However, this leads to *content-agnostic* placement of I frames at the beginning of each segment, resulting in encoding overheads. To avoid this, recent solutions started practicing variable length video split [5, 22]. This method considers appropriate placement of I frames when splitting the video. This requires fewer I frames while achieving lower bitrate without quality degradation. Specifically, we follow a video split method from Netflix that detects visually noticeable changes in subsequent frames to split a video into segments [5]. To detect such changes, we estimate how different each frame is from its previous one. If the difference is above the predefined threshold value, we start a new segment. Note that we can adjust the fixed-length based adaptive bitrate algorithms to the variable length as proposed in [4].

The variable-length video split allows each video segment to be represented by its I frame as all other frames in the segment are visually similar. By using the high level features from each I frame, we cluster the I frames into groups with similar features to construct a micro SR model for each cluster. To extract the high level features from I frames, we use a *Variational Autoencoder* (VAE), a well-known *generative* and *unsupervised* neural network model [13].

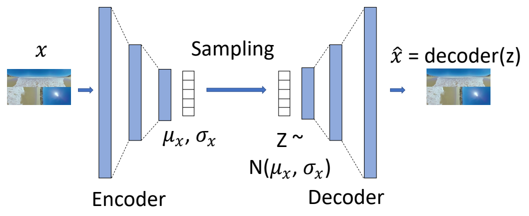


Figure 3: Variational Autoencoder Structure

As shown in Figure 3, the VAE consists of an encoder and a decoder neural network, aiming to minimize the reconstruction error between the input (x) and the reconstructed data ($\hat{x}$). Through iterative training process, it learns two mappings: one that maps the input images (x) into the distribution of the latent space represented as the mean μ_x and covariance σ_x matrices (encoder), and the other that maps the latent vector back to the reconstruction ($\hat{x}$) of the original images (decoder). A subtle but important point to note here is that we train both encoder and decoder, but we use only encoder to get the latent features which are then fed as input to the clustering algorithm. The training loss function of the VAE consists of a *reconstruction* term and a *regularization* term as below:

$$L(x, \hat{x}, \mu_x, \sigma_x) = c||x - \hat{x}||^2 + KL[N(\mu_x, \sigma_x), N(0, 1)], \qquad (1)$$

where KL and N denote the Kulback-Leibler divergence [14] and normal (Gaussian) distribution, respectively. As in the normal AutoEncoder, the reconstruction loss term ($||x - \hat{x}||^2$) aims to make the $\hat{x}$ as close to x as possible. Whereas, the regularization term ($KL[N(\mu_x, \sigma_x), N(0, 1)]$) aims to regularize the organization of the latent space by making the distributions returned by the encoder close to a standard normal distribution. By adding the regularization term to the loss function, we can enforce the VAE to organize the latent space in such a way that *two close latent vectors in the latent space return similar reconstructions once decoded, capturing the high level features efficiently.* Unlike the VAE, normal autoencoders do not guarantee that two close latent vectors in the latent space return similar reconstructions once decoded. In other words, two adjacent vectors in the latent space built by a normal autoencoder can return totally different reconstructions, which is against what we want to achieve through feature extraction and segment clustering processes.

3.1.2 Video Segment Clustering. Once we extract latent features using the VAE, we leverage the K-means algorithm [17], unsupervised machine learning algorithm to group visually similar segments together. K-means algorithm divides a set of N samples into disjoint K clusters, each of which can be represented by a *centroid.* To group similar data points together while separating the dissimilar, the K-means algorithm iteratively searches the centroids that minimize the *inertia* that measures how internally coherent clusters are. Given the latent vectors from the feature extraction step, the K-means algorithm groups visually similar video segments together, while separating the different ones as far as possible as shown in Figure 4. Even though the K-means algorithm generally works well and fast by heuristically solving the clustering problem, there is a downside that it can converge to the local optimum. Thus, to land on the global optimum in our clustering problem, we use the global K-means algorithm [15] instead of the original K-means algorithm.

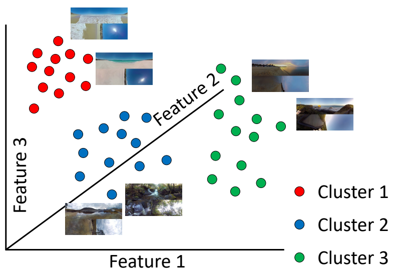


Figure 4: Video Segment Clustering

When using the K-means clustering, a general challenge is to decide the optimal hyperparameter K that can affect the overall clustering performance. A commonly used metric, called, silhouette coefficient [21] can be used to find the optimal K. The silhouette coefficient measures how far away a data point is from other clusters (separation) as well as how similar a data point is to its own cluster (cohesion). A high silhouette coefficient indicates that a data point is well matched to its own cluster and separated from neighboring clusters. Normally, silhouette coefficient alone can be sufficient to decide the optimal K in the normal clustering problem. To decide the optimal K, we iteratively calculate silhouette coefficients increasing K and select the K with the maximum silhouette coefficient as the optimum K. Figure 5 shows that the optimal number of clusters can be 16, at which we achieve the maximum silhouette coefficient.

However, relying only on silhouette coefficient can result in sub-optimal clustering performance for dcSR because of the underlying constraints such as the *SR model size* and *minimum SR performance*. As we deploy K number of micro models instead of one large model for an entire video, the overall size of the micro models should not exceed that of the large model. Such a constraint limits the range of K, the number of micro models we can deploy for a video.

We tackle the above challenge by finding the size of the *minimum working model* that not only achieves the smallest micro model size, but also enables comparable SR performance to the large model. We achieve this by profiling all possible configurations (e.g., convolution filter size) of the model architecture empirically. Note that the configuration of a minimum working model can vary for each video as each video has different visual feature distribution. Therefore, we find the *minimum* configuration and decide the range of possible K for each video separately. The detailed description on how to find the minimum working configuration can be found in in Appendix (§A.1). Formally, the K value that achieves the maximum silhouette coefficient can be decided as follows:

$$k^* = \arg\max_{k \in K} SC(K), \tag{2}$$

$$1 \leq k \leq \frac{|M_{big}|}{|M_{min}|}, \tag{3}$$

where $SC(k)$ and K denote the silhouette coefficient given k number of clusters and a set of possible k, respectively. $|M_{big}|$ and $|M_{min}|$ represent the size of one large model and that of the minimum working model, respectively.

3.1.3 Training Micro SR Models. After we cluster the video segments, we construct a micro SR model corresponding to each cluster. We extract the I frames of all segments in each cluster and train a model by feeding a low quality version of I frames with their

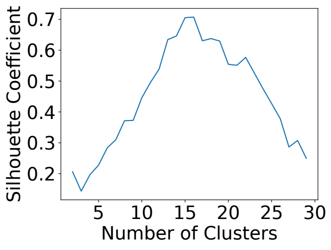


Figure 5: Optimal Number of Clusters

corresponding high quality frames as ground-truth. To train micro models, dcSR adopts an enhanced deep super resolution network (EDSR) [16] that stacks residual blocks to enhance the SR performance, and follows similar procedure for training.

3.2 Client-side dcSR

On the client-side, we integrate dcSR into a video decoder to achieve two key goals: 1) after decoding and enhancing I frames, the rest of the frames (P and B) need to reflect the enhanced I frame quality, and 2) we want to avoid redundant model downloads by caching models for future I frame quality enhancement.

3.2.1 SR Integration into Video Decoder. Figure 6 shows the workflow of client-side dcSR. As shown, we add dcSR as one additional layer of process to the existing video decoder process. Once the client receives a video segment and the corresponding micro SR model, the video decoder in the client decodes the I frame of the video segment and temporarily stores it in the decoded picture buffer (DPB) to be used to decode the subsequent P and B frames. Note that we pause the remaining decoding process for the subsequent P and B frames to apply the SR to the I frame in the DPB. Since the I frame in the DPB is in a YUV format by default, dcSR converts the format of the frame into a RGB format which is the type that a micro SR model accepts. After the color conversion, dcSR enhances the quality of the I frame in the DPB using the micro SR model. Before resuming the decoding process, dcSR converts the enhanced I frame back into a YUV format which is required for the remaining decoding process. After the conversion, dcSR resumes the decoding process by referring the P and B frames to the I frame.

3.2.2 Micro SR Model Caching. As explained above, our dcSR client downloads multiple micro SR models for the entire video, instead of a single large model. When deploying multiple micro SR models, we exploit the intermittent usage of each micro model through long-term temporal correlations across video segments to avoid redundant model downloads. For example, a segment in the start of the video can belong to the same cluster of another segment after a few minutes, and both segments can use the same model trained for the corresponding cluster. To exploit this, we cache models downloaded for earlier segments to avoid downloading the duplicate models.

Figure 7 shows a walk-through example of our model caching. Initially, dcSR downloads a video *segment 0* and *model 0*. After finishing the SR process for *segment 0*, dcSR stores *model 0* in its cache. When downloading *segment 1*, dcSR checks the corresponding model label for *segment 1*. As *segment 1* requires *model 1* and the cache does not have it, dcSR downloads *model 1*. For *segment 2*, dcSR does not request a new micro model as it already has *model*

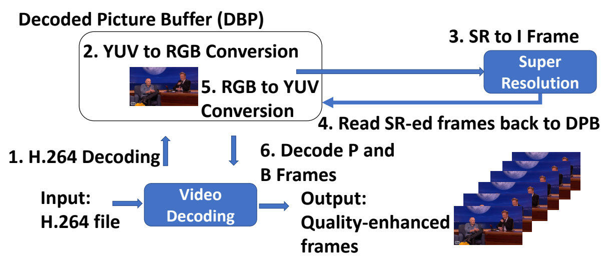


Figure 6: Workflow of Client-side dcSR

1 in its cache. When downloading *segment 3 and 6*, dcSR fetches *model 2 and 3* respectively as its cache does not have them.

Algorithm 1 describes the flow of the micro model caching process. V, C, $HashMap^L$, and $HashMap^M$ represent a list of video segment labels, a cache, a hashmap that maps a video segment label to a model label, and a hashmap that maps a model label to the corresponding model respectively, where n is the number of video segments. Given a video segment label v_i as an input, $HashMap^L$ returns the corresponding micro model label, L. Similarly, given a model label L, $HashMap^M$ returns the corresponding micro model. For each video segment represented by v_i, dcSR checks if the corresponding model (L) for v_i is in C (line 3-4). If model L is not in C, dcSR downloads it and updates its cache (line 5-6).

Algorithm 1: Model Caching and Fetching

```
Input :$V, HashMap^L, HashMap^M$
1 $C \leftarrow \emptyset$;
2 for $v_i \in V$ do
3     $L = HashMap^L[v_i]$;
4     if $L \notin C$ then
5         $C \leftarrow C \cup L$;
6         $HashMap^M[L] = DOWNLOAD(L)$
7     end
8     $Model = HashMap^M[L]$
9 end
```

4 EVALUATION

We implement dcSR on top of H.264 video codec and integrate with FFMPEG [9], an open-source multimedia framework for real-time streaming. On the server-side, SR model training process runs in Python using Tensorflow [10]. On the client-side, a SR model runs in Python along with FFMPEG built in C. We refer to it as SR-FFMPEG and release it for the community for further research in this direction [2].

To evaluate dcSR, we compare it with two SR methods: NAS [27] and NEMO [26]. In both these studies, one large SR model is trained with all the video frames in each video, and is downloaded in the beginning of the video streaming. While NAS applies SR on every video frame, NEMO applies SR on a few selected frames for faster inference. In our evaluation, NEMO is simplified to apply SR only to I frames for fair comparison with dcSR. We evaluate the performance of dcSR and benchmarks on 6 representative videos from different genres from YouTube, following the recommended encoding settings [28]. The average length of videos is 754 seconds ($\approx$ 12 minutes).

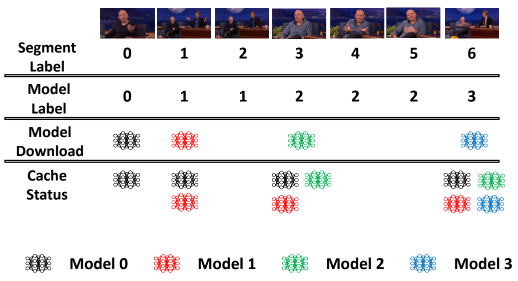


Figure 7: Micro Model Caching in Client dcSR

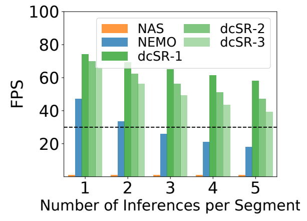
Our evaluation focuses on the following— **1)** can dcSR achieve a real-time video quality enhancement on high resolution videos (e.g., HD, FHD and 4K) across different devices? **2)** What is the power consumption of SR inference? **3)** What is the trade-off between the quality and bandwidth?

Real-time SR Inference. To evaluate the inference speed, we use three classes of devices (mobile-grade, laptop, and desktop). We show the results from mobile-grade device (Jetson Xavier NX [19]) for brevity. Additional results are in Appendix (§A.2). We evaluate dcSR with three distinct configurations that are deployed over 6 videos: dcSR-1, dcSR-2, and dcSR-3, increasing the model complexity from 1 to 3. To provide the detailed information about those configurations, dcSR-1, dcSR-2, and dcSR-3 are composed of 4, 12, and 16 ResBlocks, each of which has 16 convolution filters. Note that there can be multiple I frames in a segment in a practical setting in order to avoid the quality drift. Hence we evaluate the inference speed against enhancing i.e., SR inferencing multiple frames for a given segment. This shows that dSR can be used in both variable as well as constant length video segmentation.

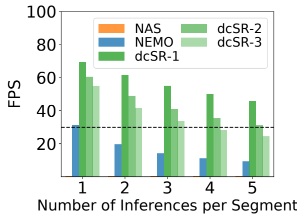
Figure 8(a)-(c) compares the average inference time of dcSR with NAS and NEMO on the Jetson platform, a mobile-grade device. The horizontal dashed line represents the original video FPS (frames per second) requirement which each method should meet. The X-axis is the number of inferences made in each segment. Note that dcSR and NEMO apply SR only to a few frames in a segment, while NAS does to every frame. Thus, the number of inferences per segment on the X-axis is only for dcSR and NEMO, while NAS has the number of video frames per segment as the number of inference per segment. To evaluate the practical FPS, we consider both the video decoding latency and the inference latency.

The key takeaway from those figures is that dcSR can meet the real-time 30 FPS requirement for all three resolutions in its lowest configuration (dcSR-1). While NEMO achieves 30 FPS under few instances for 720p resolution, it shows significantly low FPS for 1080p. For both 720p and 1080p resolutions, NAS achieves less than 1 FPS because of its bulky model complexity trained for the entire 12 minute video. Interestingly, NAS and NEMO cannot even run for 4K resolution because of running out of memory. On the other hand, when the number of inference in a segment is 1, dcSR achieves the target 30 FPS in its lowest configuration without running out of memory. Besides, dcSR achieves at least 5 FPS in a higher configuration.

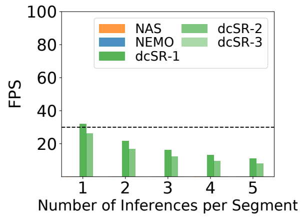
Power consumption during SR inference. SR models are very power-hungry compared to traditional video codecs. As optimizing the power consumption is essential for mobile devices, we measure the power consumption of dcSR by monitoring the power rails and



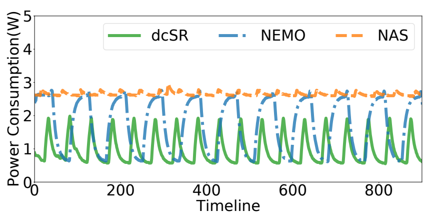
(a) 720p (HD)



(b) 1080p (FHD)

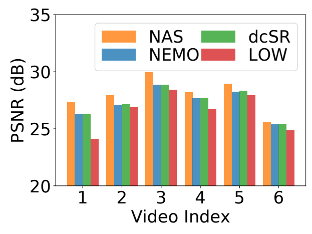


(c) 4K

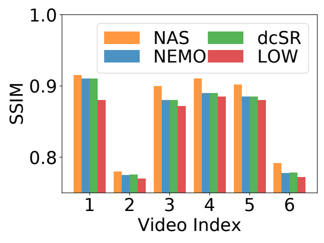


(d) Power Consumption (1080p)

Figure 8: Inference rate (a-c) and power consumption (d) of dcSR on Jetson Xavier NX (a Mobile-grade device).



(a) PSNR



(b) SSIM

Figure 9: Quality Comparison

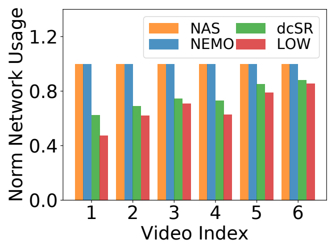


Figure 10: Network Usage Comparison

carrier board on the Jetson board using the recommended guide from Nvidia [20].

Figure 8(d) shows the power consumption of dcSR-1 compared with NAS and NEMO. Among all methods, dcSR consumes the least power (up to 2W) during the inference. The periodic power spikes for NEMO and dcSR are due to the periodic SR inference, while those for NAS are consistently high at 2.8W because it infers all the frames. In total, dcSR saves 1.4× and 2.9× of energy compared to NEMO and NAS, which shows a significant potential for running dcSR models on mobile devices.

Video quality vs. bandwidth usage. Recall that the challenge with the video-specific SR models is that we have to download the model (s) along with the video, which leads to quality and bandwidth trade-off. To this end, we compare the video quality and bandwidth usage of dcSR with NAS and NEMO.

To evaluate the quality enhancement and bandwidth saving, we generate low quality videos by setting constant rate factor (CRF) to 51 in FFMPEG. The CRF scale ranges between 0 and 51, where 0 is lossless, and 51 is the worst quality. Note that there is no bandwidth throttling in this evaluation since we do not do end-to-end streaming experiments. Instead, we compare how much bandwidth is needed with dcSR and existing methods.

Figure 9 shows the enhanced video quality of dcSR in terms of PSNR and SSIM [25] metrics. Over all 6 videos, dcSR achieves similar quality with NEMO, with both of them having no more than 1dB PSNR and 0.05 SSIM loss compared to NAS. According to the experimental results from [25], the SSIM values in our experimental results show relatively small variance in mean opinion scores, which can make a loss of 0.05 in SSIM acceptable.

Figure 10 shows the corresponding bandwidth usage. The Y-axis is the normalized bandwidth usage against the NAS bandwidth usage. Figure shows that dcSR requires on average 25% less bandwidth compared to NAS and NEMO. This is because dcSR being based on the data-centric AI paradigm achieves the comparable quality improvement with smaller models. Note that micro SR models need to be transmitted along with certain video segments in dcSR. This may reduce the available bandwidth for the video segments in a non-uniform fashion. However, it opens an opportunity to improve the overall QoE for ABR algorithms that use dcSR. For example, an ABR algorithm can use the decoded and super-resolved quality level as an input to trade the network and compute capacity as in [7, 27]. Finally, we also note that training dcSR micro models require significantly less training time (by 3×) compared to the large model designs followed by NAS and NEMO. This can potentially reduce the training costs significantly.

5 CONCLUSION

We have proposed a practical super-resolution approach, dcSR, that enables real time neural video quality enhancement on commodity devices. Unlike the existing approaches, dcSR follows a data-centric paradigm in designing SR models on per-video basis by exploiting the long term temporal correlations of video segments. We have demonstrated that such a design has the promise of faster inference, significantly less power consumption, and less bandwidth requirement with a comparable quality relative to the existing methods. We envision that dcSR will be a practical option toward pervasive SR for future coming 4K and 8K streaming service. As our future work, we plan to design a user study to evaluate how our approach affects users' perceived quality improvement.

ACKNOWLEDGEMENTS

This work is supported by Korean National Research Foundation (NRF) grant NRF-2021R1F1A1052489, and Korea Radio Promotion Association (RAPA) XR-LAB grant. The authors are grateful to the anonymous reviewers and our shepherd (Eric Rozner) for their valued comments and suggestions. We thank graduate students Binayak Das and Hangil Kang for their help in bootstrapping the project and with setting up the streaming platforms.

REFERENCES

[1] Namhyuk Ahn, Byungkon Kang, and Kyung-Ah Sohn. 2018. Fast, accurate, and lightweight super-resolution with cascading residual network. In *Proceedings of the European Conference on Computer Vision (ECCV '18)*. 252–268.

[2] Duin Baek. 2021. dcSR: SR-FFMPEG. https://github.com/barrelo89/dcSR. (2021).

[3] Duin Baek, Hangil Kang, and Jihoon Ryoo. 2020. SALI360: Design and Implementation of Saliency Based Video Compression for 360° Video Streaming. In *Proceedings of the 11th ACM Multimedia Systems Conference (MMSys '20)*. 141–152.

[4] Luca Bedogni, Marco Di Felice, and Luciano Bononi. 2017. Dynamic segment size selection in HTTP based adaptive video streaming. In *2017 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. 665–670.

[5] Netflix Technology Blog. 2018. Optimized shot-based encodes: Now Streaming. https://netflixtechblog.com/optimized-shot-based-encodes-now-streaming-4b9464204830. (2018).

[6] C. Scott Brown. 2018. https://www.androidauthority.com/qualcomm-snapdragon-mobile-npu-896223/. (2018).

[7] Mallesham Dasari, Arani Bhattacharya, Santiago Vargas, Pranjal Sahu, Aruna Balasubramanian, and Samir R. Das. 2020. Streaming 360-Degree Videos Using Super-Resolution. In *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*. 1977–1986.

[8] DeepLearningAI. 2021. A Chat with Andrew on MLOps: From Model-centric to Data-centric AI. https://www.youtube.com/watch?v=06-AZXmwHjo. (2021).

[9] FFMPEG. 2021. FFMPEG. https://www.ffmpeg.org/. (2021).

[10] Google. 2021. Tensorflow. https://github.com/tensorflow/tensorflow. (2021).

[11] Zheng Hui, Xiumei Wang, and Xinbo Gao. 2018. Fast and accurate single image super-resolution via information distillation network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR '18)*. 723–731.

[12] ITU. 2019. H.264 : Advanced video coding for generic audiovisual services. https://www.itu.int/rec/T-REC-H.264. (2019).

[13] Diederik P Kingma and Max Welling. 2014. Auto-Encoding Variational Bayes. (2014). arXiv:stat.ML/1312.6114

[14] S. Kullback and R. A. Leibler. 1951. On Information and Sufficiency. *Ann. Math. Statist.* 22, 1 (03 1951), 79–86.

[15] Aristidis Likas, Nikos Vlassis, and Jakob J. Verbeek. 2003. The global k-means clustering algorithm. *Pattern Recognition* 36, 2 (2003), 451 – 461.

[16] Bee Lim, Sanghyun Son, Heewon Kim, Seungjun Nah, and Kyoung Mu Lee. 2017. Enhanced Deep Residual Networks for Single Image Super-Resolution. *IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW '17)* (2017), 1132–1140.

[17] S. Lloyd. 1982. Least squares quantization in PCM. *IEEE Transactions on Information Theory* 28, 2 (1982), 129–137.

[18] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. 2017. Neural Adaptive Video Streaming with Pensieve. 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 (SIGCOMM '17)*. 197–210.

[19] NVIDIA. 2021. JETSON Xavier NX. https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-xavier-nx/. (2021).

[20] NVIDIA. 2021. NVIDIA Jetson Linux Driver Package Software Features. https://docs.nvidia.com/jetson/l4t/. (2021).

[21] Peter J. Rousseeuw. 1987. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *J. Comput. Appl. Math.* 20 (1987), 53–65.

[22] S. Schwarzmann, T. Zinner, S. Geissler, and C. Sieber. 2018. Evaluation of the Benefits of Variable Segment Durations for Adaptive Streaming. In *2018 Tenth International Conference on Quality of Multimedia Experience (QoMEX '18)*. 1–6.

[23] Kevin Spiteri, Rahul Urgaonkar, and Ramesh K. Sitaraman. 2020. BOLA: Near-Optimal Bitrate Adaptation for Online Videos. *IEEE/ACM Transactions on Networking* 28, 4 (2020), 1698–1711.

[24] Gary J Sullivan, Jens-Rainer Ohm, Woo-Jin Han, and Thomas Wiegand. 2012. Overview of the high efficiency video coding (HEVC) standard. *IEEE Transactions on circuits and systems for video technology* 22, 12 (2012), 1649–1668.

[25] Zhou Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. 2004. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing* 13, 4 (2004), 600–612.

[26] Hyunho Yeo, Chan Ju Chong, Youngmok Jung, Juncheol Ye, and Dongsu Han. 2020. NEMO: Enabling Neural-Enhanced Video Streaming on Commodity Mobile Devices. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking (MobiCom '20)*. Article 28, 14 pages.

[27] Hyunho Yeo, Youngmok Jung, Jaehong Kim, Jinwoo Shin, and Dongsu Han. 2018. Neural Adaptive Content-aware Internet Video Delivery. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI '18)*. 645–661.

[28] Youtube. 2019. Upload videos: Recommended upload encoding settings. https://support.google.com/youtube/answer/1722171?hl=en. (2019).

n_{RB} \ n_f	4	8	12	16	20
4	0.276	0.454	0.633	0.813	1.0
8	0.344	0.566	0.789	1.000	1.2
16	🟩 0.599	0.988	🟩 1.4	🟩 1.8	2.2
32	1.6	2.7	3.7	4.9	🟥 5.8
64	5.6	9.3	13.0	16.7	20.5

Table 1: Model Size (MB) Over Different Model Configurations (n_f: # of filters, n_{RB}: # of ResBlocks)

A APPENDIX

A.1 Finding The Minimum Working Model

To find the minimum working model, we conduct tests on hyper-parameters, and observed that the number of convolution filters and the number of ResBlocks in EDSR [16] are the major factors to affect both the model size and SR performance.

Based on the observation, we run the configuration grid search incrementing those hyperparameter values, which generates a configuration table. Table 1 shows how a micro model changes in its size by those hyper-parameters. Given the configuration table of model sizes, we iteratively build and evaluate a micro model with a configuration in the table in an ascending order of model size.

Once we find the configuration that achieves the comparable SR performance to the big model over the I frames, we have it as the minimum working model configuration. Note that the big model is trained on all the video frames in a video, whereas a micro model is trained only on the I frames in a video. Besides, the image feature based K-means clustering lowers the variance in the training data. Thus, a micro model can achieve comparable SR performance to the big model even with simpler model architecture. The distinct configurations used for 6 videos are marked green. The big model configuration is marked red.

Since we utilize the overfitting or memorization of neural networks to enhance the video quality, we can theoretically expect that a model can memorize training data better or more easily given fewer training data. To confirm it in practice, we tested the overfitting or memorization performance by increasing training data size. To isolate the overfitting performance from different initial weight values, we initialized a micro model with the same weight for different training data sizes. As shown in Figure 11, the training loss increases as the amount of data a micro model has to *memorize* or *overfit* increases. Since training data and test data for each micro model are identical in our system, the training loss represents how well a micro model can enhance the quality of video content. As expected, the smaller data given to a model, the better visual quality enhancement it can achieve. Thus, we can expect that each micro model may memorize assigned training data better, as the average amount of training data per micro model decreases. Besides, the similarity between the training dataset for each model may further simplify the training process.

A.2 Inference Speed on Laptop and Desktop

To evaluate the inference speed in laptop and desktop, we use a laptop equipped with an Intel i7-7700HQ CPU and GEFORCE GTX

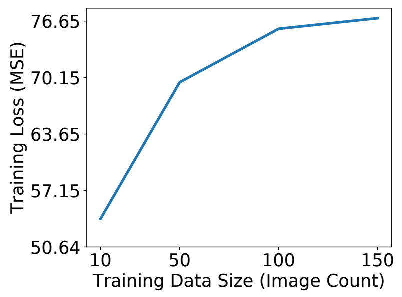


Figure 11: Training Loss over Different Training Data Size (8 filters / 8 ResBlocks)

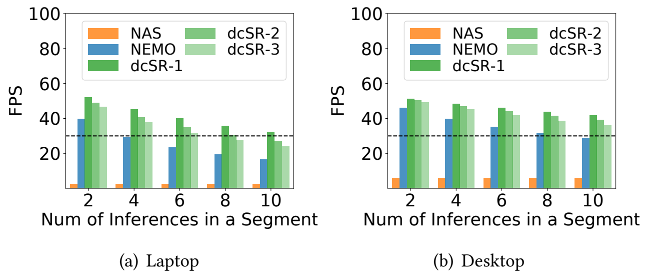


Figure 12: Inference Rate in Laptop and Desktop

1060 GPU, and a desktop equipped with an Intel i7-8700 and RTX 2070 GPU. Figure 12 compares the inference time of dcSR on 4K videos with those of NAS and NEMO. Since the performance of the other methods on HD and FHD videos were already covered in the literature [26, 27], we evaluate their performance on 4K videos.

While NEMO achieves 30 FPS under few instances, dcSR can meet the real-time 30 FPS requirement regardless of device type and the number of inference in a video segment. As in the mobile-grade device, NAS fails to achieves the required FPS. Since the laptop and desktop used in the experiment are high-performance devices, it is expected that the FPS will decrease for commodity devices, which will make it harder for NAS and NEMO to meet the FPS requirement.