NeuroScaler: Neural Video Enhancement at Scale

Hyunho Yeo Hwijoon Lim Jaehong Kim Youngmok Jung Juncheol Ye Dongsu Han
KAIST

ABSTRACT

High-definition live streaming has experienced tremendous growth. However, the video quality of live video is often limited by the streamer's uplink bandwidth. Recently, neural-enhanced live streaming has shown great promise in enhancing the video quality by running neural super-resolution at the ingest server. Despite its benefit, it is too expensive to be deployed at scale. To overcome the limitation, we present NeuroScaler, a framework that delivers efficient and scalable neural enhancement for live streams. First, to accelerate end-to-end neural enhancement, we propose novel algorithms that significantly reduce the overhead of video super-resolution, encoding, and GPU context switching. Second, to maximize the overall quality gain, we devise a resource scheduler that considers the unique characteristics of the neural-enhancing workload. Our evaluation on a public cloud shows NeuroScaler reduces the overall cost by 22.3× and 3.0-11.1× compared to the latest per-frame and selective neural-enhancing systems, respectively.

CCS CONCEPTS

• **Information systems** → **Multimedia streaming**; • **Computing methodologies** → **Computer vision**; • **Computer systems organization** → **Real-time system architecture**;

KEYWORDS

live streaming, super-resolution, deep neural networks

ACM Reference Format:
Hyunho Yeo, Hwijoon Lim, Jaehong Kim, Youngmok Jung, Juncheol Ye, Dongsu Han. 2022. NeuroScaler: Neural Video Enhancement at Scale. In *ACM SIGCOMM 2022 Conference (SIGCOMM '22), August 22–26, 2022, Amsterdam, Netherlands.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3544216.3544218

1 INTRODUCTION

The demand for live streaming has rapidly grown over the last decade—live video traffic is expected to take up 17 percent of Internet traffic by 2022 [12]. Current live streaming infrastructure relies on two key pieces: 1) at the ingest side, the streamer uploads a video to the media server using the low-latency streaming protocols [46, 47]; and 2) at the distribution side, the clients run an adaptive bitrate (ABR) algorithm [4, 15, 64] to select the highest quality video that can be streamed in real time.

Unfortunately, traditional live streaming falls short of consistently delivering high-quality video (e.g., 4K/8K) because the ingest video quality critically depends on the streamer's uplink bandwidth [106, 112]. When the ingest path becomes congested, the entire downstream video quality suffers directly [56, 69]. However, video quality is the most important factor that affects user engagement in live streaming [65, 77]. For example, more than 50% of live viewers abandon a stream when video quality suffers more than 90 seconds [28]. The churn of these viewers can in turn greatly harm the revenue of live streaming providers [51, 58].

Recent advances in neural-enhanced streaming [39, 50, 74, 85, 91, 98] show great promise in enhancing the ingest video quality by utilizing computations at a media server. When the ingest video quality suffers, the media server recovers high-quality video by running *end-to-end* neural enhancement that consists of 1) decoding a low-quality stream, 2) applying a super-resolution deep neural network (DNN) to the stream, and 3) encoding the super-resolved outputs. This delivers a dramatic quality improvement in the downstream video.

However, neural enhancement is too costly to support commercial-scale live streaming. For example, Twitch [70] supports more than 100,000 concurrent live streams [59]. Applying end-to-end neural enhancement in this setting requires tens of thousands of GPUs, which costs over $169,000 per hour on a public cloud (§3). Our cost breakdown shows that both video super-resolution and encoding are expensive. Neural super-resolution requires 100-1000× more computations compared to a DNN used for discriminative tasks [88], and video encoding is up to 3.3× slower than super-resolution.

In this paper, we aim to develop a scalable and resource-efficient neural-enhancing framework, which reduces the operating cost by an order of magnitude and efficiently scales out to a cluster of computing instances. Building such a framework involves a number of non-trivial challenges:

- First, existing video super-resolution methods are expensive. Running neural super-resolution on a per-frame basis is infeasible. The state-of-the-art selective super-resolution [101] reduces the overhead by applying a DNN to selective frames and by reusing the outputs for other frames. However, it is not designed for live video and involves expensive offline computation.
- Second, super-resolved video must be (re-)encoded in real time for live streaming. However, traditional video codecs [25, 26, 33] are computationally expensive in compressing high-resolution videos (4K/8K), often becoming a bottleneck in end-to-end neural enhancement (§3).
- Lastly, to maximize the overall quality improvement on a large number of streams and computing instances, resources must be optimally allocated for each stream at each instance. However, typical resource schedulers cause imbalances in per-instance and per-stream load, which in turn leads to noticable quality degradation (§3).

SIGCOMM '22, August 22–26, 2022, Amsterdam, Netherlands

ACM ISBN 978-1-4503-9420-8/22/08…$15.00
https://doi.org/10.1145/3544216.3544218

This paper presents NeuroScaler, the first work that identifies and solves the main computational bottlenecks of neural-enhanced live streaming. NeuroScaler addresses the challenges above by introducing new system designs:

End-to-end optimizations: We design novel algorithms that significantly reduce the costs of video super-resolution, video encoding, and GPU context switching. First, to accelerate video super-resolution, it extends selective super-resolution but presents an *online algorithm* that chooses frames to apply super-resolution (i.e., anchor frames) using codec-level information in real time. In contrast to prior work [101] that requires O(|frame|) neural inference, our algorithm selects anchor frames *without* any inference while providing the same quality gain. Second, we devise a *hybrid video codec* that is specialized for encoding selective super-resolution outputs. Instead of re-encoding every output frame, the hybrid codec reuses the original input video and compresses only the super-resolved anchor frames using an image codec. Then, the video and the enhanced anchor frames are packaged together in a single stream and delivered to clients. Compared to traditional video codecs, the hybrid codec reduces the encoding cost by an order of magnitude, while achieving similar compression efficiency.

Efficient resource scheduling: We introduce a novel *anchor-aware resource scheduler*, which considers the unique characteristics of anchor frames for efficiently utilizing a computing cluster. First, the quality gains of anchor frames are heterogeneous. Thus, to maximize the overall quality, the scheduler runs at a centralized server to select the most beneficial anchor frames across all streams. Next, the computing overheads of anchor frames are heterogeneous across streams and can change over time. Therefore, to accurately balance the load among computing instances at scale, the scheduler dynamically estimates the number of anchor frames each instance can process and forward them as scheduled.

We evaluate NeuroScaler with a full system implementation. Our evaluation using real-world videos and GPU cloud instances shows that NeuroScaler greatly improves the system throughput, resource efficiency, and cost-effectiveness of end-to-end neural enhancement. Compared to the latest per-frame and selective neural enhancement, NeuroScaler respectively improves the processing throughput by 10× and 2.5-5× and reduces the overall cost by 22.3× and 3.0-11.1×.

In summary, we make three key contributions:

- **Scalable neural enhancement.** NeuroScaler is the first work to identify and solve the key bottlenecks of live neural enhancement (§3).
- **End-to-end optimizations.** NeuroScaler proposes zero-inference anchor frame selection (§5.1), hybrid codec (§6.1), and context switching optimizations for supporting multiple, concurrent live streams (§6.2), greatly reducing the end-to-end neural enhancement cost.
- **Efficient resource allocation.** NeuroScaler introduces a novel anchor-aware scheduler (§5.2) that maximizes the overall quality gain across concurrent live streams.

This work does not raise any ethical issues.

2 BACKGROUND

Live streaming consists of the ingest and distribution side, as illustrated in Figure 1. First, at the ingest side, the streamer captures live video and uploads it to a media server. In adaptive streaming [4, 15], the streamer uploads a single video stream, and the media server transcodes it into multiple quality versions. In multi-party video conferencing [64], a broadcaster uploads multiple quality streams using simulcast [63] or scalable video codec (SVC) [48], and the media server forwards the streams to viewers. Next, at the distribution side, the client runs an adaptive bitrate (ABR) algorithm to choose/download the highest quality video under the given network bandwidth.

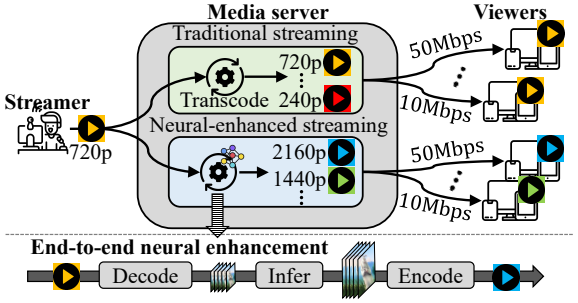


Figure 1: Traditional vs. Neural-enhanced streaming (Case for adaptive streaming)

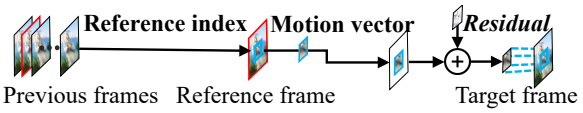


Figure 2: Selective super-resolution overview

The common limitation of traditional live streaming is that video quality is limited by the streamer's uplink bandwidth [56, 69, 106]. Even if viewers have ample network bandwidth, the limitation can deprive them of the opportunity to enjoy high-quality video.

Super-resolution is a class of techniques that produces a high-resolution image from a lower resolution counterpart. Recent studies [78, 84, 86, 89, 107] use deep neural networks (DNN) for learning the mapping from low-resolution to high-resolution and demonstrate dramatic quality improvements.

Neural-enhanced live streaming [39, 50, 74, 85, 91, 98] employs super-resolution DNNs to enhance the ingest stream, as shown in Figure 1. In contrast to traditional live streaming, the media server runs end-to-end neural enhancement. When a video arrives, it is first decoded into raw frames. Then, the DNN is applied to all or selective frames depending on a super-resolution method. The outputs are re-encoded into a video before delivering it to clients.

To provide reliable quality improvement, a content-aware DNN is commonly used for each stream, which can be also updated during live streaming. The context switching between content-aware DNNs incurs two types of overhead. First, recent DNN compilers [3, 42, 109, 110] provide optimized inference on a target accelerator, but this involves an upfront cost due to model optimization, which takes up to minutes. It applies both graph-level and operator-level optimizations [42], generating a DNN tailored for a specific target. Second, before running inference, a DNN and frames must be loaded to the device memory of a target accelerator. Because neural accelerators commonly have limited memory size (e.g., 16GB in NVIDIA T4 [41]), multiple content-aware DNNs, each requiring several GBs of memory, must be frequently swapped in/out.

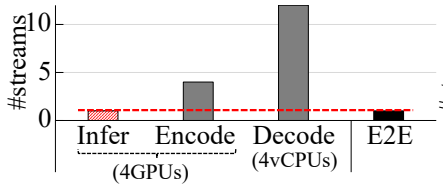


Figure 3: Per-frame SR is limited by the inference overhead.

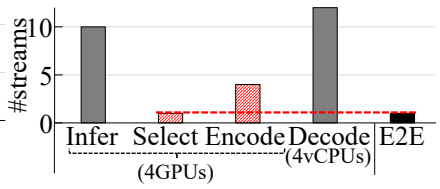


Figure 4: Selective SR is limited by the selection/encoding overhead.

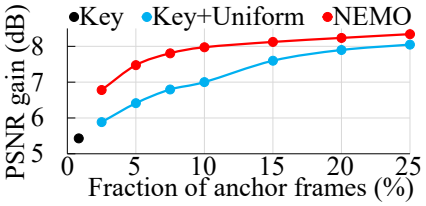


Figure 5: Naive anchor selection methods degrade the quality.

Selective super-resolution reduces the computing overhead by utilizing the temporal redundancy across video frames. Neural inference is only applied to selective frames, referred to as *anchor frames*. As shown in Figure 2, non-anchor frames are reconstructed by reusing the previous super-resolved frames guided by codec-level information (e.g., reference index, motion vector, residual). This involves lightweight bilinear interpolation and decoding and thus can be processed in real time even on mobile devices [101].

When a non-anchor frame is up-scaled by reusing the previous super-resolved frames, quality loss (compared to per-frame inference) inevitably occurs due to temporal difference. This loss accumulates across consecutive non-anchor frames, but is reset at an anchor frame. Therefore, it is important to select a beneficial set of anchor frames to maximize the quality. For this, NEMO [101] relies on the costly per-frame inference. As a result, this cannot be used for live streaming in which anchor frame selection must be done online in real time.

3 CHALLENGES

To demonstrate that end-to-end neural enchantment is prohibitively expensive and inefficient, we benchmark its cost and quality gain on a public cloud.

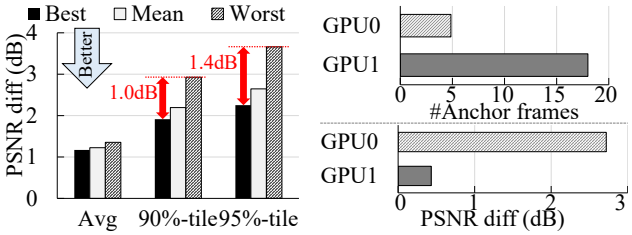
3.1 Expensive End-to-end Enhancement

End-to-end video super-resolution consists of the decode, infer, and (re)encode processes and is prohibitively expensive. To demonstrate this, we benchmark two super-resolution (SR) methods: 1) *per-frame SR* applies neural super-resolution to every frame, and 2) *selective SR* applies neural super-resolution to anchor frames (~7.5% of frames), which are chosen by the NEMO algorithm [101]. We use a GPU cloud instance [8] that has four inference-optimized NVIDIA T4 GPUs [41]. We run the "high quality" DNN from NAS [102], which up-scales a 720p, 60 fps video [19] to a 2160p version. For fair comparison, we configure both approaches achieve a similar quality by adjusting the DNN size: Original (32.39 dB in PSNR), Per-frame SR (40.12 dB), Selective SR (40.19 dB).

• Per-frame SR. Figure 3 shows the processing throughput of each process (decode, infer, and encode) run in isolation and the combined end-to-end throughput; the bars represent the number of live streams that can be processed in real time. The per-frame SR can process only a single stream in real time due to the expensive DNN inference. This costs at least $1.690 per hour per stream using the cloud GPU instance. At Twitch scale, with 100,000 concurrent live streams [59], this translates to $169,000 per hour.

• Selective SR (NEMO) shows 10× improvement in inference throughput, as shown in Figure 4. However, it requires expensive per-frame inference for offline anchor frame selection, which chooses a set of anchor frames that improves video quality the most. The anchor selection is a one time cost for on-demand video, but offline processing is not feasible for live streaming. Alternatively, simply applying neural super-resolution only to key frames (Key SR) or key frames and equally-spaced normal frames (Key+Uniform SR) eliminates the costly anchor selection process, but results in large quality degradation due to ineffective anchor frames. Figure 5 shows video quality in PSNR against the fraction of anchor frames for different selective SR methods. Compared to NEMO, 1) Key SR degrades video quality by 1.34-2.90 dB, and 2) Key+Uniform SR requires 2.5-3.0× more anchor frames to achieve the same quality.

Insight #1. *Live neural enhancement can greatly benefit from selective inference, provided that impactful anchor frames can be selected in real time.*



(a) Quality difference (b) Per-GPU statistics

Figure 6: Anchor-agnostic resource scheduling results in noticeable quality degradation.

Video encoding. Super-resolved frames must be re-encoded in real time for live streaming, but encoding high-resolution videos (4K/8K) is expensive. Even if selective inference effectively reduces the inference overhead, the end-to-end processing is still slow because video re-encoding becomes a key bottleneck. To demonstrate this, we benchmark the VP9 software encoder [33] and the H.265 hardware encoder [40] on the same instance as above.

Figure 4 compares the processing throughput of video encoding and selective inference. The hardware encoder (using on-chip GPUs) can process four 4K, 60 fps live streams, but it is 2.5× slower than selective inference. Even worse, hardware codecs are often unavailable on neural accelerators [2, 7, 9, 13]. In this case, running the software encoder [33] is 5× slower than the selective inference. Alternatively, video encoding can be offloaded to (off-chip) video codec FPGAs/ASICs [10, 35]. However, transmitting 4K/8K frames consumes too much bandwidth, even with lossless image compression [44] (e.g., 2.2-9.0 Gbps per stream) and can cause a large delay, making it difficult, if not infeasible, to support live streaming.

Insight #2. *To accelerate end-to-end neural enhancement, reducing the video encoding overhead is also critical.*

3.2 Inefficient Resource Scheduling

To maximize the overall quality improvement on a computing cluster, 1) the number of anchor frames has be carefully allocated across streams, and 2) anchor enhancement tasks must be balanced across computing instances. However, this is challenging due to the characteristics of anchor frames. First, the quality gains of anchor frames are heterogeneous. Second, the computing overheads of anchor frames are heterogeneous across streams and can change over time (according to the variation of ingest resolution). Using naive stream-level load-balancing [31, 36] causes imbalances in per-stream anchor frames and per-instance load, which in turn leads to noticeable quality degradation.

To demonstrate this, we benchmark a strawman *anchor-agnostic scheduler* that 1) allocates video streams to GPUs in a round-robin manner, 2) applying NeuroScaler's end-to-end neural enhancement (§4) per GPU. We use two servers, each with a single NVIDIA T4 GPU [41]. Five 360p and 720p streams (total of 10 streams) are up-scaled to 1080p and 2160p versions, respectively; a 720p frame is 4.2× more expensive to enhance compared to a 360p frame. The experiment is repeated 1,000 times by randomly shuffling the order of video streams. For each iteration, we measure the quality difference from per-frame super-resolution for each video chunk.

Figure 6(a) shows the average, 90%-tile, 95%-tile quality difference of video chunks for the best, mean, and worst case. The anchor-agnostic scheme fails to consistently minimize the quality difference across iterations. There is a noticeable difference between the best and worst case: 0.18 dB (average), 1.0 dB (90%-tile), and 1.4 dB (95%-tile). This is because the anchor-agnostic scheme suffers from an *imbalance* in per-stream anchor frames, which results from an *imbalance* in per-instance load. To demonstrate this, Figure 6(b) shows the per-GPU statistics for the worst case, where the 720p and 360p streams are placed on GPU0 and GPU1, respectively. In GPU0, a few anchor frames are selected per chunk (4.86 on average), which leads to large quality difference (2.72 dB). In contrast, in GPU1, a larger number of anchor frames are chosen per chunk (18.0), but the quality gain per anchor frame greatly decreases (0.17 dB). Anchor frames are under-/over-selected in GPU0 and GPU1, respectively.

Using other load balancing schemes [22, 31, 36] for neural inference (e.g., hashing, least connection, shorted expected delay) cannot resolve the problem because they do not manage resources in an anchor-frame granularity.

Insight #3. *To attain high resource efficiency on a computing cluster, we need a resource allocation scheme that considers the unique characteristics of neural video enhancement.*

4 NEUROSCALER OVERVIEW

Goal. Motivated by the limitations of neural-enhanced live streaming, our goal is to reduce the cost of end-to-end neural enhancement and maximize overall quality gain on a computing cluster.

Scope. Neural-enhanced live streaming involves additional DNN training and neural enhancement cost. In this paper, we scope our work to reduce the end-to-end neural enhancement cost because it is significantly more expensive than the training cost. For example, LiveNAS [85] uses three GPUs for the enhancement, while only one GPU for the training. The training cost can be further reduced by sharing a super-resolution DNN across similar videos [71, 85, 97].

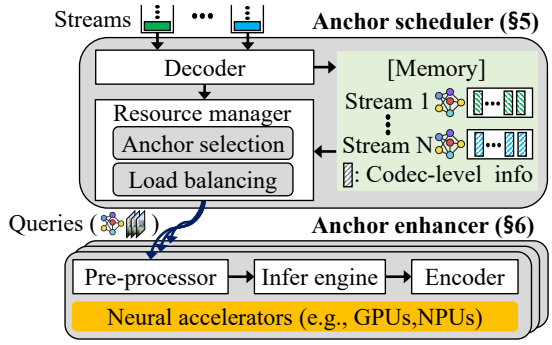


Figure 7: NeuroScaler overview

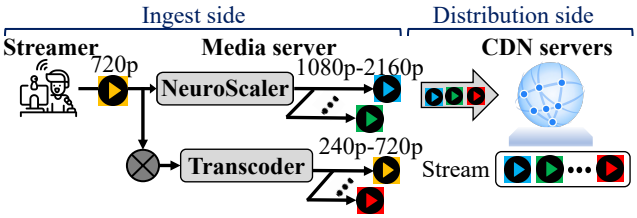


Figure 8: NeuroScaler deployment model

Overview. NeuroScaler takes an ingest stream and a DNN pair and outputs a high-resolution stream; the DNN can also be dynamically updated through online learning as in LiveNAS. Figure 7 shows the overall workflow of NeuroScaler that consists of the *anchor scheduler* (§5) and the *anchor enhancers* (§6). The scheduler decodes ingest streams and selects the most beneficial anchor frames across the streams. Then, for each stream, the DNN and the selected anchor frames are forwarded to the enhancers, which are equipped with neural accelerators. The enhancers pre-process a DNN, apply the DNN to the anchor frames, and re-encode the super-resolved outputs. All the processes in the scheduler and the enhancer are pipelined and parallelized to maximize the overall processing throughput.

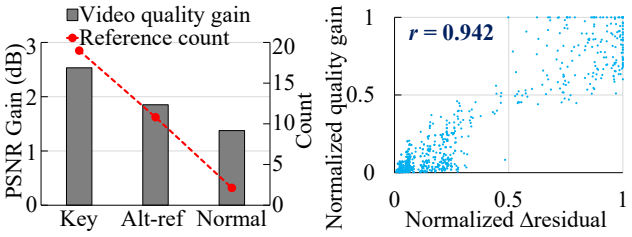
Deployment scenario. Figure 8 illustrates the deployment model of NeuroScaler for adaptive streaming. When a low-quality stream (e.g., 360p or 720p) is uploaded to a media server, NeuroScaler produces a higher-resolution stream using real-time super-resolution. Lower-resolution versions (e.g., 240p-720p) are also created from the ingest stream using a traditional transcoding pipeline [66]. By using NeuroScaler, clients can now watch high-resolution video even when the ingest path becomes congested. Deploying NeuroScaler on video conferencing is similar to the above process, but does not require multi-resolution transcoding.

5 ANCHOR SCHEDULER

This section describes NeuroScaler's anchor frame selection algorithm (§5.1) and resource management modules (§5.2) that utilizes the algorithm.

5.1 Zero-inference Anchor Frame Selection

Problem & Goal. To run selective super-resolution on live content, both anchor frame selection and enhancement must be processed in real time. To this end, we develop a *zero-inference* algorithm 1) that selects anchor frames without any neural inferences, 2) while



(a) Frame type (b) Residual

Figure 9: Key observations on anchor frames

offering comparable quality to that of the algorithm running the per-frame inference.

Insight. Our key observation is that the benefit of using an anchor frame can be estimated by information from a video codec, without actual neural inferences. In particular, the benefit critically depends on frame type and residual:

Frame type: Frames inside a compressed video have dependencies in the form of a directed graph. There are special types of video frames with a high degree of reference, and using them as anchor frames delivers larger quality improvements. For VP8/9 and AV1 codecs [5, 62], two types of frames belong to this category: 1) *key frames* are the first frames of a group of pictures (GOP) and 2) *alternative reference frames* are invisible frames only used for inter-frame compression. To demonstrate this, we measure the quality improvement on video chunks [68] when using an anchor frame per frame type. Figure 9(a) indicates that key and alternative reference frames have a higher reference count (within a graph of frames) and deliver 1.2 dB, 0.5 dB higher quality in PSNR compared to normal frames, respectively.

Residual: Reusing super-resolution results produces quality loss due to residual (i.e., temporal difference) between frames (§2). Thus, the gain of an anchor frame is proportional to the amount of loss it reduces; but measuring this loss requires running actual selective super-resolution. Instead, our key observation is that this loss can be approximated by the amount of residual an anchor frame reduces, which can be easily calculated as in Equation 1. To demonstrate this, we measure the reduced residual and the video quality gain when using an alternative reference frame as an anchor frame. Both values were normalized to between 0 and 1 within each video chunk [19]. Figure 9(b) shows that there is a high correlation between quality gain and the reduced residual; the Pearson correlation coefficient [80] (r) is 0.942.

Design. Based on the observations above, we develop a *zero-inference anchor frame selector* that leverages codec-level information (e.g., frame type, residual) to select beneficial anchor frames. Figure 10 illustrates the overall workflow:

① **Divide:** The selector divides frames into groups based on their type for each stream. There are a total of three groups in which G_{key}, G_{altref}, and G_{normal} have key frames, alternative reference frames, and other frames, respectively. The groups have priorities in selecting anchor frames in the order of G_{key}, G_{altref}, and G_{normal}.

② **Estimate:** The selector estimates the benefit of using an anchor frame, referred to as *anchor gain*, for the frames in G_{altref} and G_{normal}. The anchor gain is calculated based on the accumulated residual, using Algorithm 1 explained later in this section. For the frames in G_{key}, we assume they have equal anchor gain as they do

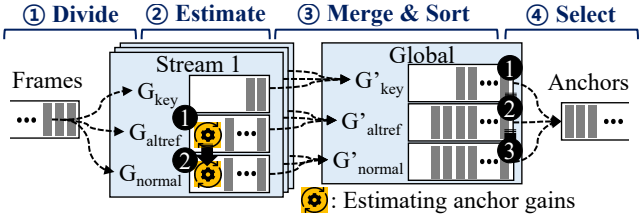


Figure 10: Zero-inference anchor selection algorithm

not have an effect on accumulated residual. This does not harm the overall performance because there are only a few key frames, and all of them are commonly selected as anchor frames.

③ **Merge & Sort:** The selector merges per-stream groups into global groups, where G^{global}_{key}, G^{global}_{altref}, and G^{global}_{normal} contain key frames, alternative reference frames, and normal frames of all streams, respectively. Frames in the same group are sorted according to the anchor gain.

④ **Select:** The selector iteratively chooses anchor frames from the sorted global groups starting from G^{global}_{key} to G^{global}_{altref} and G^{global}_{normal}. To meet the real-time constraint, NeuroScaler measures the latency of DNNs once and selects the maximum number of anchor frames whose total latency is less than the available computing time.

Estimating anchor gain. In Step ②, NeuroScaler uses residual to estimate anchor gain using Algorithm 1 in Appendix A. The algorithm first calculates accumulated residuals across frames (Algorithm 1, line #2). It then iteratively selects the most beneficial frame and estimates its anchor gain. For each iteration, it calculates the amount of residual each frame reduces (line #6-8) as follows:

$$\begin{aligned}\Delta\mathrm{Res}(F[i]) &= \sum_j \Big(\underbrace{\mathrm{Res}(F[j])}_{F[i]\neq\text{Anchor}} - \underbrace{\mathrm{Res'}(F[j])}_{F[i]=\text{Anchor}} \Big) \\ &= \sum_{j=i}^{k-1} \big(\mathrm{Res}(F[j]) - (\mathrm{Res}(F[j]) - \mathrm{Res}(F[i]))\big) \\ &= (k-i) \times \mathrm{Res}[i]\end{aligned} \tag{1}$$

where $\mathrm{Res}(F[j])$ is accumulated residual of the j th frame, and k is the index of the closest frame at which the residual resets. The residual is cleared at either a key frame or a frame whose anchor gain was estimated in previous iterations. If such frames do not exist within the given frames, the algorithm predicts that the residual resets at the key frame of the next video chunk. Next, it selects the frame that reduces the residual the most and sets its anchor gain as the amount of reduced residual (line #12,13). Lastly, it updates the accumulated residual of each frame to reflect the impact of the chosen frame (line #14). To quickly estimate the anchor gain, the total residual pixel value is approximated as the size of an encoded residual frame. This has a minimal impact on quality (=Δ0.05 dB in PSNR) because both values are highly correlated.

5.2 Anchor-aware Resource Management

Goal & Challenge. Our goal is to maximize the overall quality improvement when processing a large number of streams. For this, we want NeuroScaler to optimally allocate computing resources for each stream, while balancing the load. However, this is challenging because the quality gain and the computing overhead of anchor frames are heterogeneous across streams and can change over time.

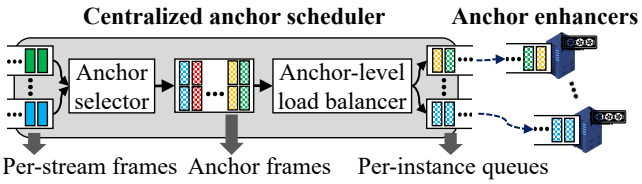


Figure 11: Anchor-aware scheduler overview

In particular, the stream-level load balancers in §3 miss two key opportunities for maximizing the quality. First, the scheduler selects locally optimal anchor frames because it runs end-to-end neural enhancement per instance. Second, the scheduler suffers from an imbalance in per-stream anchor frames across instances. The computing overhead of stream processing is non-deterministic; it dynamically changes according to several factors such as anchor frames per stream and available resources. This leads to an imbalance in per-instance load, which in turn causes an imbalance in per-stream anchor frames.

Design. To resolve the challenge, we make two design choices. First, to choose an optimal set of anchor frames, we run a *global* anchor frame selection across all streams. Second, to mitigate the two types of imbalance, we distribute the load to instances at an *anchor-frame* granularity, in which the overhead of anchor frames can be accurately estimated. To this end, we propose an *anchor-aware resource scheduler* consisting of two main modules, as illustrated in Figure 11.

① **Global anchor frame selector** runs at a centralized server to select the most beneficial anchor frames across all video streams. In particular, it periodically runs the zero-inference algorithm (§5.1), choosing the largest number of anchor frames that can be processed in real time on a computing cluster as follows:

$$\max_{N} \left(\sum_{i=1}^{N} T_{DNN}(AF[i]) \leq T_{intv} \times M \right)$$

where AF is anchor frames sorted by the anchor gain; T_{DNN} is the DNN latency; T_{intv} is the anchor frame selection interval, which is a configurable parameter; and M is the number of computing instances in the cluster. In this study, we use T_{intv} = 666 ms (40 frames for 60 fps video) unless otherwise noted. Such global anchor frame selection allows us to mitigate an imbalance in per-stream anchor frames.

In general, using the longer scheduling interval can increase quality by selecting more impactful anchor frames, as shown in Figure 29 in Appendix C. However, this incurs higher end-to-end latency; thus the interval must be adjusted according to the application requirement.

② **Anchor-level load balancer**, after selecting anchor frames, dynamically balances the loads among the computing instances at an anchor-frame granularity. To do so, the resource scheduler divides the selected anchor frames into per-instance groups, where frames in each group are forwarded to and processed at the corresponding computing instance. To meet the real-time constraint, the resource scheduler assigns the maximum number of anchor frames for each group, whose total latency is below the anchor frame selection interval (T_{intv}). Such anchor-level load balancing allows us to mitigate an imbalance in per-instance load.

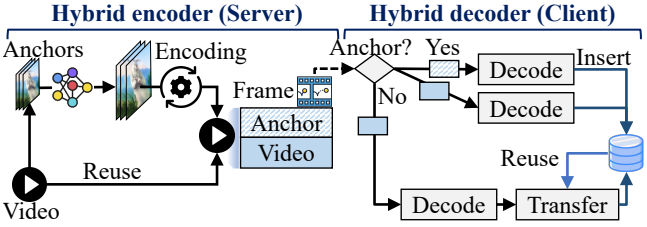


Figure 12: Hybrid codec: overall workflow

Trade-off policies. There is a trade-off between quality and throughput. More streams can be processed per instance by reducing the number of anchor frames per stream, which results in lower quality due to the reduced computations. NeuroScaler provides two different trade-off policies:

The *cost-effective* policy operates at the knee point of the curve between cost and quality gain; refer to Figure 16 in §8.1. Increasing the cost beyond this point returns the marginal quality gain while decreasing the cost greatly degrades the quality. To find this knee point, an operator needs to profile a trade-off between the fraction of anchor frames and quality gain over representative videos. With this policy, resource auto-scaling is required to support all incoming live streams; thus, NeuroScaler dynamically provides the number of computing instances needed as $ceil(\frac{T_{DNN}(AF)}{T_{intv}})$, where $T_{DNN}(AF)$ is the latency of anchor frame enhancement.

The *latency-sensitive* policy selects the same fraction of anchor frames as above, but the scheduling interval is set to 66 ms (4 frames for 60 fps video) to satisfy the delay requirement (200 ms) of live video conferencing [52].

6 ANCHOR ENHANCER

This section describes NeuroScaler's hybrid codec (§6.1) and GPU context switching optimizations (§6.2).

6.1 Hybrid Video Encoding

Problem & Goal. End-to-end neural enhancement requires (re-)encoding, but traditional video encoders (e.g., VPx [28], H.26x [9, 10]) are computationally expensive and become a main computational bottleneck (§3). Thus, we aim to develop a lightweight codec that can compress high resolution streams at least as fast as our selective inference, such that they do not become a bottleneck, while offering comparable quality to the existing codecs. This lightweight codec allows us to process end-to-end neural enhancement at the speed of selective super-resolution.

Insight. Our key observation is that non-anchor frames can be bypassed when (re-)encoding the output of selective super-resolution. This is because non-anchor frames can be easily reconstructed at the client side even with commercial mobile devices without significant overhead [101]. Such by-passing would allow us to encode the target video very fast. In addition, as the non-anchor frames are directly sent at low resolution without up-scaling, this further leaves room, in terms of bandwidth, for encoding the anchor frames at high quality. Motivated by these observations, we devise a *hybrid codec*, which is specialized for encoding selective super-resolution outputs. Figure 12 illustrates the overall workflow of the hybrid codec.

Hybrid encoding (server-side). In contrast to traditional encoders, the hybrid encoder utilizes both video and image codecs in

a synergistic way. It reuses the input video stream and compresses only super-resolved anchor frames using an image codec, while offloading non-anchor frame reconstructions to clients. Each anchor frame size is equally set to meet the bitrate constraint in live streaming. Next, the hybrid encoder packages the encoded video and super-resolved anchor frames in a single file, which is later delivered to the clients. A new header attached to each frame contains the frame type (i.e., anchor or non-anchor frame) and the super-resolved frame for an anchor frame.

The hybrid encoder has two key advantages. First, it reduces the computing overhead by 78.6-235.8× compared to a video encoder (§8.2). This is because the hybrid encoder applies a lightweight image codec, which is ~6.25× cheaper than a video encoder, only to a few anchor frames (5-10% of all frames). Second, because anchor frames are sparsely spread apart within a video chunk, applying an image codec to a few frames has a minimal effect on compression efficiency; the Bjontegaard rate difference (BD-rate) [73] improves by 6.69% when using the hybrid encoder compared to re-encoding a video using a traditional encoder (§8.2).

Hybrid decoding (client-side). The hybrid-encoded stream is decoded at the client, as shown in Figure 12. When a compressed frame arrives, the hybrid decoder first checks if the current frame is an anchor frame. If it is, the decoder decodes the super-resolved frame and caches it in memory. Otherwise, the codec reuses previous super-resolved frames to up-scale the current frame by leveraging codec-level information (§2). Decoding a hybrid-encoded stream incurs minimal overhead. Our evaluation (§8.2) shows that the hybrid decoder slightly increases energy consumption by 18% compared to a traditional decoder.

6.2 Optimizing GPU Context Switching

We aim to efficiently execute super-resolution DNNs on inference-optimized frameworks (e.g., TensorRT [42], TVM [3]). However, in neural-enhanced live streaming, naively using these frameworks severely degrades the speed due to two types of GPU context switching overhead (§2).

Problem (DNN update). Model optimization (§2) can make inference faster, but it commonly takes several seconds to minutes depending on the DNN size. Thus, translating the benefit of this optimization to the live streaming context is challenging. Because a DNN can be updated online, the optimization must be done in real time.

Model pre-optimization. To reduce the optimization latency, we rely on our key observation that the model optimization result is less relevant to its actual weight values. Based on this, we propose a model pre-optimization scheme in which the optimization takes place offline and occurs just once. Before live streaming begins, NeuroScaler takes a randomly initialized "mock" DNN and pre-optimizes it on a target accelerator. Next, when a DNN needs optimization during live streaming (§4), our system generates the optimized version using the mock DNN, which was optimized offline. This involves replacing the parameters of the mock DNN with those of the target DNN. This scheme reduces the latency from tens of seconds to several milliseconds (§8.2).

Problem (DNN loading). Multiple DNNs/frames arrive at each anchor enhancer per scheduling interval (§5.2); thus device/host memory must be frequently allocated/freed. Such overhead is comparable to that of neural inference when it comes to high-resolution (4K/8K), because several MBs/GBs of host/device memory are required per DNN, respectively.

Memory pre-allocation. To avoid the frequent memory allocation overhead, we pre-allocate host/device memory and construct a memory pool when a program launches. In Appendix A, we describe the memory management scheme that considers the characteristics of neural super-resolution. This scheme makes the overhead negligible regardless of the requested memory size.

7 IMPLEMENTATION

NeuroScaler is implemented upon commercial frameworks including libvpx (v1.10) [33], TensorRT (v8.0.3) [53], libjpeg-turbo (v2.1.2) [32], Kakadu H2JTK (v8.2.1) [30], and gRPC (v1.42.0) [24]. NeuroScaler consists of ~10.1K lines of code.

Decoder. We extend libvpx, which is a reference software implementation of VP9. The original decoding API only outputs a decoded visible frame, but we need codec-level information (e.g., residual, frame type) and invisible frames for anchor frame selection and enhancement, respectively. Therefore, we modify the decoding API (`vpx_codec_get_frame`) to additionally return this information. Next, we parallelize per-stream decoding on multiple CPU threads. Since frames within the same stream have dependency for decoding, we allocate a dedicated thread for each stream.

8 EVALUATION

We evaluate NeuroScaler by answering the following questions.

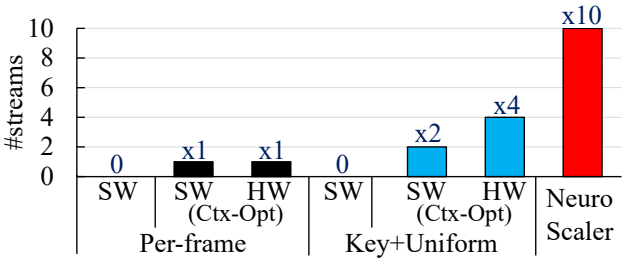
- Does NeuroScaler effectively improve the end-to-end processing throughput?
- How does each design component of NeuroScaler contribute to the overall performance?
- Does NeuroScaler effectively maximize the overall quality of multiple streams?

Hardware. Experiments were conducted on AWS EC2 `g4dn` instances [8] that have Intel 2.5 GHz Cascade Lake CPUs and NVIDIA T4 GPUs. Table 1 in Appendix B presents their specifications and prices. We ran neural inference on the GPUs and other processes (e.g., decoding, encoding, anchor frame selection) on the CPUs.

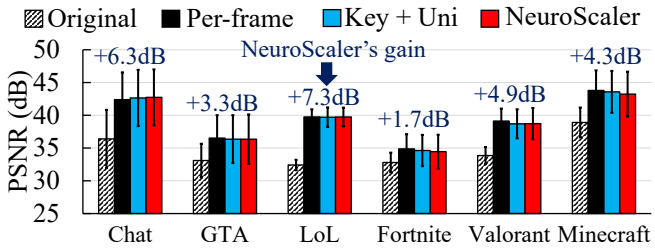
Video. We setup raw videos and encoded them for ingest and distribution as follows:

Setup: We used videos from the top six most-watched content categories on Twitch in 2021 [34]. Because Twitch does not provide 2160p, 60 fps video, we downloaded videos from Youtube with the same content. For each category, we sorted videos by view count and selected the top video that supports 2160p, 60 fps and is at least 20 minutes long [16–21].

Ingest: We used the VP9 codec following the Youtube Live settings [69]. Group of picture (GOP), frame rate, and bitrates were respectively configured as 2 seconds, 60 fps, and {0.7, 4.125, 6.75, 35.5} Mbps (for {360, 720, 1080, 2160}p video). Detailed codec parameters we use are specified in Appendix B. Unless otherwise noted, the first 20 minutes of 720p videos were used. Note that 720p is most widely used for broadcasting in Twitch, whereas 2160p is rarely used [1, 11].



(a) End-to-end processing throughput



(b) Quality gain of neural enhancement

Figure 13: NeuroScaler greatly improves end-to-end processing throughput compared to the baselines under similar SR quality. (SW/HW: Software/Hardware codec, Ctx-Opt: Context switching optimization)

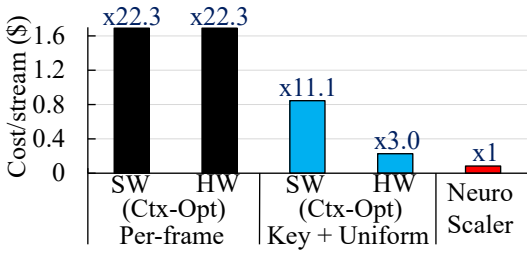


Figure 14: End-to-end processing cost based on cost-effective cloud GPU instances

Method	#streams
Key+Uniform SR	0
Ctx-Opt	2
Ctx-Opt + Anchor-Sel	2
Ctx-Opt + Hybrid-Enc	4.33
Ctx-Opt + Hybrid-Enc + Anchor-Sel	10

Figure 15: End-to-end throughput breakdown of NeuroScaler

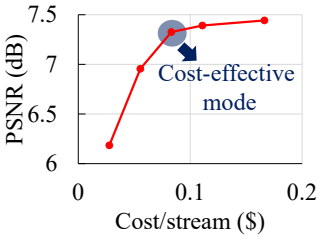


Figure 16: Trade-off btw cost and quality in NeuroScaler

Distribution: We compressed super-resolved frames as follows. First, for per-frame encoding, the software VP9 codec [33] and the hardware H.265 codec [40] were used with the same configuration as above; both codecs show a similar level of compression efficiency. Next, for hybrid encoding, the software JPEG2000 codec [30] was used with the quantization parameter (QP) as Table 2 in Appendix B.

Super-resolution DNN. We used the "high-quality" DNN from NAS [102], which has 8 residual blocks, 32 channels, and a scale factor of 3. To emulate the benefit of online learning as in LiveNAS [85], a content-aware DNN was trained per video with the first 10 minutes and tested with the last 10 minutes. However, NeuroScaler is orthogonal to how the super-resolution network is trained. The cost-effective mode (§5.2) was used for NeuroScaler unless otherwise noted.

8.1 End-to-End Performance

To demonstrate NeuroScaler delivers significant improvement in scalability, we compare NeuroScaler with two end-to-end neural enhancment baselines:

- **Per-frame baseline** applies a DNN to every frame and encodes the output using the traditional video codec; the state-of-the-art neural-enhanced live streaming (LiveNAS [85]) adopts this approach.
- **Selective baseline** applies a DNN to every key frame *and* equally-spaced frames (Key+Uniform), while using the same codec as above.

For fair comparison, we configure the baselines, as best as possible, to achieve the same quality as NeuroScaler. For this, we adjust the DNN size for the per-frame baseline and the number of anchor frames for the selective baseline. Table 3 in Appendix B shows the configuration per content.

Throughput improvement. Figure 13(a) illustrates the average end-to-end neural enhancement throughput across the six videos[1]. The maximum number of streams that can be processed in real time was measured on the `g4dn.12xlarge` instance that has 48 vCPUs and 4 GPUs. NeuroScaler significantly improves the end-to-end processing throughput by 10× and 2.5-5× compared to the per-frame and the selective baseline, respectively. Without the GPU context switching optimization (§6.2), both baselines cannot process even a single stream in real time. With the optimization, the end-to-end performances of the per-frame and the selective baseline are constrained by the inference and encoding overhead, respectively.

Quality gain. Figure 13(b) shows the original and neural-enhanced video quality in PSNR; the error bars represent one standard deviation from the average. The quality was measured between the raw video and the compressed (super-resolved) video. NeuroScaler consistently delivers large quality improvements by 1.65-7.33 dB (4.63 dB on average) compared to the original video. The absoulte PSNR of NeuroScaler ranges from 34.5 dB ("Fortnite") to 43.2 dB ("Minecraft"); Note that the quality difference between the baselines and NeuroScaler is minimal (0.215 dB on average). In Appendix C, we demonstrate that NeuroScaler's gain is also significant in VMAF [55] in Table 5, which is a widely-used video quality metric; we present the snapshots of video samples in Figures 30, 31, and 32.

Cost saving. To compare the end-to-end processing cost, we select the most cost-effective computing instance for each approach. Figure 14 compares the per-stream cost[1]. It shows that NeuroScaler significantly reduces the cost by 22.3× and 3.0-11.1× compared to the per-frame and selective baselines (with the GPU context switching optimization), respectively. Table 4 in Appendix C lists the most cost-effective instance type and the number of instances needed per 100 streams for each approach.

[1]We omit the standard deviation in the figure as the number of real-time streams or the processing cost is the same across contents.

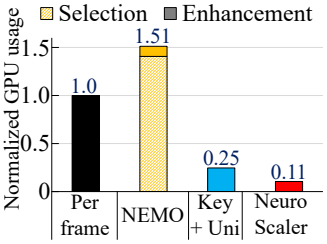


Figure 17: SR inference resource usage

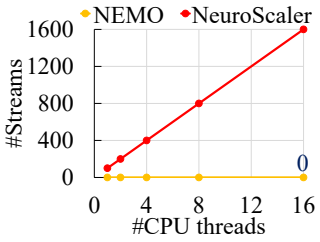


Figure 18: Anchor frame selection throughput

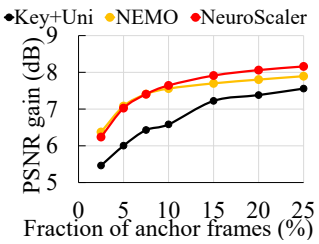


Figure 19: Anchor frame efficiency in quality gain

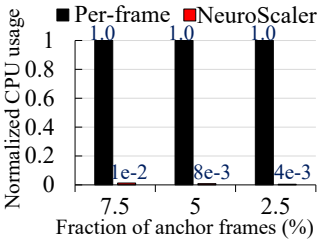


Figure 20: Video encoding resource usage

NeuroScaler's large cost reduction (up to 22.3x) comes from the fact that NeuroScaler greatly reduces both CPU and GPU usage, which enables NeuroScaler to run on more economic instance types. Table 7 in Appendix C quantifies this resource saving. The reduced GPU usage is due to selective inference with the zero-inference algorithm (§5.1) and the context switching optimization (§6.2). The reduction in CPU usage comes from the use of the hybrid codec (§6.1). As a result, NeuroScaler runs on more economic instances, such as `g4dn.xlarge` (4 vCPUs per GPU), in contrast to `g4dn.12xlarge` (12 vCPUs per GPU) used in Figure 13(a).

Contributions by individual components. Figure 15 illustrates the throughput breakdown of NeuroScaler using the `g4dn.12xlarge` instance that has four GPUs. We tested multiple versions of NeuroScaler by selectively applying the zero-inference anchor selection (§5.1), the hybrid encoding (§6.1), and the context switching optimization (§6.2). The result shows that all components significantly contribute to the improvements in the number of streams. The context switching optimization enables processing two streams in real time (1st→2nd row), while the vanilla selective SR fails to support even a single live stream. The number of streams is increased by 2.16× due to the hybrid codec (2nd→4th row) and by 2.30× due to the zero-inference anchor selection (4th→5th row).

Cost-effective mode. The default cost-effective mode (§5.2) balances the quality and the cost. To demonstrate this, we measured the quality gain against the cost per stream for the "League of Legends" video [19], as shown in Figure 16. Increasing the cost (33.3-100%) beyond this mode returns the marginal quality gain (0.07-0.12 dB). Otherwise, reducing the cost (33.3-66.6%) greatly sacrifices the quality by 0.37-1.14 dB.

End-to-end latency. NeuroScaler can support live adaptive streaming and video conferencing using the cost-effective and latency-sensitive policy, respectively. To demonstrate this, we benchmark the former and latter policy on the `g4dn.xlarge` and `g5dn.2xlarge` instance; the latter has a NVIDIA A10 GPU. Table 8 in Appendix C shows the end-to-end latency and its breakdown. First, the cost-effective mode incurs delay of 669 ms on average (±338 ms standard deviation). Since traditional live streaming (e.g., Twitch) incurs dealy of several seconds [27, 60], the addition latency from neural enhancement has a minimal impact. Second, the latency-critical mode produces delay of 90.8 ms on average (±25.8 ms), which satisfy the delay requirement (200 ms) of video conferencing [52]. The delay can be further reduced, if needed, by decreasing the anchor frame selection interval (T_{intv}) and/or running DNNs on more advanced accelerators [37, 43] which provide lower latency.

8.2 Component-wise In-depth Analysis

We provide an in-depth performance analysis of individual system components.

NeuroScaler's anchor selector (§5.1) with selective inference greatly reduces the cost of super-resolution compared to the existing approaches. This is because this selector can choose beneficial anchor frames very fast.

Resource saving: Figure 17 shows the average GPU usage for neural super-resolution over the six videos; all approaches deliver similar quality (average Δ|PSNR| = 0.215 dB). The GPU usage was normalized by that of the per-frame SR. NeuroScaler reduces the GPU usage by 9.48×, 14.33×, and 2.33× compared to the per-frame, NEMO-selective, and uniform-selective SR, respectively. In contrast, the NEMO-selective SR even increases the GPU usage by 57% compared to the per-frame counterpart because of the anchor frame selection overhead; it requires per-frame inference with a larger DNN to achieve the similar quality.

Throughput: Figure 18 illustrates the anchor frame selection throughput against the number of CPU threads. NeuroScaler can process 100 streams per CPU thread (with 4.13 ms delay), whereas NEMO cannot run it on CPU in real time.

Quality: Figure 19 shows the PSNR gain against the fraction of anchor frames for the "League of Legends" video [19]. NeuroScaler 's anchor frames deliver comparable quality gain to those of the NEMO: up to 0.27 dB higher (fraction ≥ 7.5), up to 0.14 dB lower (fraction < 7.5). Compared to selecting key and equally-spaced frames as anchor frames, NeuroScaler can achieve the same quality with 2.5-3× fewer anchor frames.

NeuroScaler's hybrid codec (§6.1) greatly reduces the cost compared to re-encoding at the ingest server, while maintaining high compression efficiency. At the same time, it incurs minimal decoding overhead at the client.

Resource saving: Figure 20 shows the average CPU usage for the hybrid video encoding (JPEG2000) and the per-frame video encoding (VP9); both approaches deliver similar quality (average Δ|PSNR| =0.01 dB). These encoders were tested on various fractions of anchor frames, and the CPU usage was normalized by that of the per-frame encoding. NeuroScaler reduces the CPU usage by 78.6-235.8× compared to the per-frame encoding. The speedup becomes higher as the fraction of anchor frames reduces.

Throughput: Figure 21 illustrates the processing throughput against the number of CPU threads. NeuroScaler can process 81 streams with 16 CPU threads, while the traditional codec can encode only a single stream.

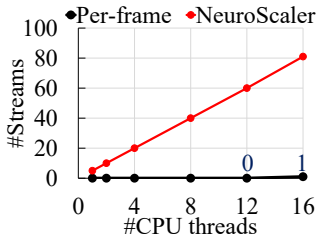


Figure 21: Video encoding throughput

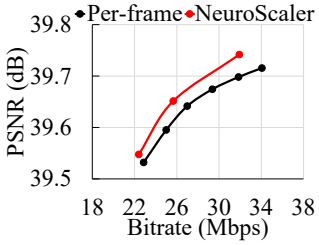


Figure 22: Video encoding efficiency (Distribution-side)

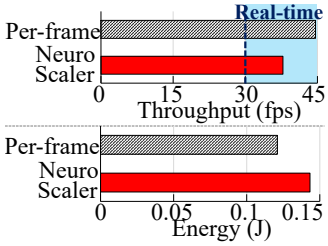


Figure 23: Video decoding overhead on a smartphone [67]

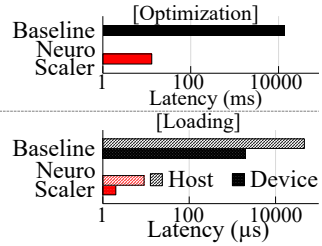
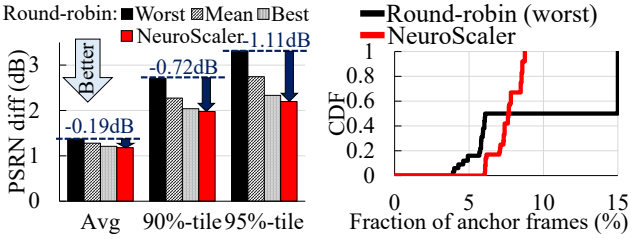


Figure 24: GPU context switching overhead



(a) Quality difference (b) #anchor frames

Figure 25: The benefits of the anchor-level load balancer

Component	Decoder	Resource manager
Instance type	c6i.32xlarge	c6i.32xlarge
Latency	2.65ms	4.13ms
#streams	768	12800
Cost (per stream)	¢0.311	¢0.0186

Figure 26: NeuroScaler's anchor scheduler scalability

Compression efficiency: Figure 22 illustrates the quality against the bitrate using the same video as above. The NeuroScaler's codec delivers comparable quality to the per-frame encoding (VP9). Overall, the hybrid codec slightly increases BD-rate [73] by 6.69%, which is a widely-used quality metric for compressed video, while processing 78.6-235.8× faster than the per-frame codec as shown above.

Decoding overhead: We benchmark the throughput and per-frame power consumption of the hybrid and traditional decoder on a smartphone [67], which has the Qualcomm Snapdragon 855 processor [49]. Figure 23 shows the throughput and energy consumption when decoding a 4K, 30 fps video [68] on the mobile CPU (4 threads). The hybrid decoding satisfies the real-time constraint, while slightly increasing the energy consumption by 18% compared to the traditional decoding (VP9). We expect that the energy consumption can be reduced as the current implementation is an un-optimized prototype which leaves room for improvement; e.g., it decodes anchor frames twice (with JPEG2000 and VP9), but such redundancy can be removed. Using the desktop class-CPU (Intel i9-9900K [29]), the hybrid decoder can process a 4K, 60 fps live video with a single thread, as shown in Table 6 in Appendix C.

NeuroScaler' GPU context-switching optimization (6.2) greatly reduces the overhead of GPU context switching, as shown in Figure 24. First, the model pre-optimization method reduces the latency of DNN compilation from 137 s to 13 ms. Next, the memory pre-allocation method reduces the latency of loading data (e.g., DNN, frames) to GPU memory from 19.9-46.5 ms to several microseconds. These two optimizations improve the inference throughput by 2.79× compared to running an unoptimized DNN using PyTorch [45].

NeuroScaler's resource scheduler (§5.2) maximizes the quality gain of selective inference. To demonstrate this, we measure the quality difference from per-frame super-resolution for each video chunk. We compare the quality with the *anchor-agnostic* scheduler that 1) allocates video streams to computing instances in a round-robin way, and 2) selects/enhances anchor frames as NeuroScaler per computing instance. We use eight servers [8], each with a single NVIDIA T4 GPU. 18 360p and 720p streams (total of 36 streams) were up-scaled to 1080p and 2160p versions, respectively; this is the maximum number of streams the cost-effective mode can process. We repeated the experiments 1,000 times by randomly shuffling the order of video streams.

Quality improvement: Figure 25(a) shows the average, 90%-tile, and 95%-tile quality difference of video chunks, respectively; note that NeuroScaler's quality gain compared to the original videos is 4.77 dB on average. The result shows that NeuroScaler's scheduler effectively minimizes the quality difference and achieves a consistent performance. Compared to the baseline, NeuroScaler reduces the average, 90%-tile, and 95%-tile quality difference by up to 0.19 dB, 0.71 dB, and 1.11 dB, respectively. In contrast, the anchor-agnostic approach shows a large variation in quality difference across iterations.

NeuroScaler's gain comes from mitigating an imbalance in per-stream anchor frames by the global scheduling of anchor frames and anchor enhancement tasks. To highlight this, we compare the fraction of anchor frames between NeuroScaler and the worst-case baseline, as shown in Figure 25(b). In contrast to NeuroScaler that balances anchor frames across streams, the baseline suffers from an imbalance in per-stream anchor frames, which results from imbalance in per-instance load (§3). In particualr, it 1) under-selects anchor frames for 15% of streams, which results in large quality difference, and 2) over-selects anchor frames for 50% of streams, which results in small quality gain per anchor frame.

8.3 Scalability

Scheduler. NeuroScaler's anchor scheduler (§4), which consists of the decoder and the resource manager, can operate at scale. Figure 26 shows the cost and the latency of each module using the `c6i.32xlarge` instance [6] that has 128 vCPUs. The scheduler overall costs ¢0.329 per stream, while the decoder and the resource manager incurs 2.65 ms and 4.13 ms delay, respectively. Here, video decoding and per-stream anchor gain estimation (Step ② in Figure 10) are the main computational bottlenecks, but both of them can be parallelized on multiple CPUs.

Case study (Twitch [57]). We estimate the operating cost for running NeuroScaler on a Twitch-scale service, which has 100,000 concurrent live streams. Figure 27 quantifies the cost using AWS EC2

Module	Instance	#instances	Cost
Scheduler	c6i.32xlarge	139	$332 per hour
Enhancer	g4dn.xlarge	33,334	$7,566 per hour
All	-	-	$7,898 per hour

Figure 27: NeuroScaler's cost for a Twitch-scale service

instances [6, 8]. NeuroScaler's neural enhancement costs $7,898 per hour, which is 21.3× less expensive than the per-frame counterpart as used in LiveNAS [85]. The anchor scheduler and enhancer accounts for 4.3% and 95.7% of the overall cost, respectively.

9 DISCUSSION

Codec neutrality. The design of NeuroScaler is codec neutral. First, the zero-inference anchor frame selection (§5.1) can be applied to any types of video codecs, since the algorithm uses features that are generally supported in video codes: 1) multiple tiers of frames varying on the degree of reference and 2) residual produced by inter-frame compression. For example, to support the H.26x codecs [25, 26], we only need to replace the VPx-/AV1-specific frame groups (G_{key}, G_{altref}, and G_{normal} in Figure 10) with the H.26x-specific groups (G_I, G_P, and G_B). Second, the hybrid encoding (§6.1) can accommodate any combinations of video codecs and images codecs. It does not rely on features that are available only for specific codecs. Lastly, the GPU context switching optimization (§6.2) and the anchor-aware resource scheduling (§5.2) are agnostic to video codecs.

Joint optimization. The performance of NeuroScaler can be further improved by considering the interdependencies among selective inference, DNN training, and video encoding, which we leave as future work. First, a DNN is applied only to anchor frames, while each anchor frame delivers a different amount of benefit. Therefore, a DNN can be trained more efficiently by 1) targeting anchor frames (instead of randomly sampled frames), and 2) allocating training time across anchor frames considering their benefit. Second, the benefit of anchor frames largely depends on frame dependencies within a video, but traditional video encoders construct such dependencies without considering anchor frames. If a video is encoded in an anchor-aware manner, the number of anchor frames could be decreased while maintaining the same video quality.

10 RELATED WORK

Accelerating super-resolution. Several studies [72, 87, 88] accelerate super-resolution at an image-level or a video-level. First, blocks within the same image have heterogeneous difficulty in recovering high-resolution counterparts. Based on this, MobiSR [88] and ClassSR [87] use fewer computations on blocks that have lower difficulty, accelerating image super-resolution. These approaches can be applied to NeuroScaler to accelerate super-resolution on anchor frames. Second, video has a large amount of temporal redundancy, and super-resolution results can be reused over consecutive frames. dcSR [72] and FAST [108] apply super-resolution only to key frames, but this can greatly degrade quality on videos that contain dynamic scenes [101].

Using super-resolution on ingest streams. Live neural-enhanced streaming is gaining increasing popularity both in academia [85, 99, 100] and industry [38, 39, 54]. LiveNAS [85] runs super-resolution on live ingest streams to enhance the video quality. Runespoor [99] and CloudSeg [100] apply super-resolution to images from surveillance cameras to improve video analytics. NVIDIA Maxine [39] provides an SDK that supports super-resolution on NVIDIA GPUs, and commercial live streaming services [38, 54] integrate this SDK to improve the service quality. In NeuroScaler, we greatly improve its cost and resource efficiency.

Using super-resolution at clients. Recent studies [76, 101, 102, 105] utilize client computations to run video super-resolution. NAS [102] applies super-resolution to adaptive streaming. Parsec [76] and VoluSR/YuZu [104, 105] use super-resolution for 360-degree and volumetric video streaming, respectively. NEMO [101] enables super-resolution on commercial mobile devices. These efforts focus on improving the quality at the distribution side and are orthogonal to our approach that improves the ingest stream quality.

Improving live streaming. A large body of work has been devoted to improving live streaming. First, at the ingest side, Salsify [79] and Concerto [111] enable fast and accurate adaptation to bandwidth fluctuations by reducing the mismatch between codecs and transport protocols. Vantage [93] optimizes quality-of-experience for time-shifted viewers by selectively re-transmitting previous videos in higher quality. NeuroScaler is agnostic to ingest protocols and can support these systems. Second, at the distribution side, previous studies [83, 90, 92, 96, 103] propose better bitrate and server selection algorithms. These efforts are orthogonal to our approach as NeuroScaler targets improving ingest stream quality. Lastly, SVE [82] proposes a scalable and distributed video processing system used in Facebook. In contrast, NeuroScaler improves the scalability of neural video enhancement.

Model serving. Several efforts have been devoted to DNN-based model serving; they are orthogonal to our contributions. Clipper [75] proposes a layered architecture to ease the deployment of model serving on various ML frameworks and devices. Nexus [95], Clockwork [81], and InFaaS [94] improve the overall processing throughput when co-locating multiple models with different SLOs. All the works dynamically optimize batch size to benefit from parallel processing on modern accelerators. Clockwork further provides the predictable performance in tail latency, and InFaaS additional adapts model-variants such as model architectures and hardware platforms. These methods can be applied to NeuroScaler to co-locate various types of streams in the same cluster.

11 CONCLUSION

In this paper, we looked at the scalability issues in neural-enhanced live streaming. We found that end-to-end neural enhancement is prohibitively expensive, and typical resource schedulers are inefficient for scale-out. We presented NeuroScaler, a scalable framework for live neural enhancement that resolves the challenges by a holistic approach. First, NeuroScaler enables selective inference in live streaming by devising the zero-inference algorithm. Next, it develops the hybrid codec and the GPU context switching optimization that allow us to process end-to-end neural enhancement at the speed of selective inference. Lastly, it introduces the anchor-aware resource management that maximizes the overall quality when scaling out to a computing cluster. In our evaluation using a public cloud, we show that NeuroScaler can deliver an order of magnitude improvement in scalability.

ACKNOWLEDGMENTS

We thank our anonymous reviewers for their constructive feedback. Seungjun Lee and Seungho Baek contributed to the work in the early days of NeuroScaler. This work was supported by Institute for Information & communications Technology Promotion (IITP) grant funded by the Korea government (Ministry of Science and ICT) [No.2022-0-00117] and [No.2018-0-00693]. Dongsu Han is the corresponding author.

REFERENCES

[1] 720p Is The Preferred Broadcasting Resolution On Twitch. https://parsec.app/blog/720p-is-the-preferred-broadcasting-resolution-on-twitch-e5385a3ef08f.
[2] Amazon EC2 Inf1 Instances Website. https://cloud.google.com/tpu.
[3] Apache TVM Official Website. https://tvm.apache.org/.
[4] Apple HTTP Live Streaming Official Website. https://developer.apple.com/streaming/.
[5] AV1 Official Website . https://aomedia.org/av1/.
[6] AWS EC2 C6i Instance Official Website. https://aws.amazon.com/ko/ec2/instance-types/c6i/.
[7] AWS EC2 DL1 Instance Official Website. https://aws.amazon.com/ec2/instance-types/.
[8] AWS EC2 G4 Instance Official Website. https://aws.amazon.com/ec2/instance-types/g4/.
[9] AWS EC2 P4Instance Official Website. https://aws.amazon.com/ec2/instance-types/p4/.
[10] AWS F1 Instance Website. https://aws.amazon.com/ec2/instance-types/f1/.
[11] The Best Streaming Bitrate and Resolutions for Twitch. https://bit.ly/32rX2Pe.
[12] Cisco Visual Networking Index Report. http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.pdf.
[13] Cloud TPU Website. https://aws.amazon.com/ec2/instance-types/inf1/.
[14] CPU cores and threads per CPU core per instance type . https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/cpu-options-supported-instances-values.html.
[15] DASH Industry Forum Official Website. https://dashif.org/.
[16] Dataset: Chat Youtube video. https://www.youtube.com/watch?v=TJoAy944MoM.
[17] Dataset: Fortnite Youtube video. https://www.youtube.com/watch?v=LW4asVQsew0.
[18] Dataset: GTA5 Youtube video. https://www.youtube.com/watch?v=jYglXGpaBSg.
[19] Dataset: League of legend Youtube video. https://www.youtube.com/watch?v=Ku_q0O_kgGE.
[20] Dataset: Minecraft Youtube video. https://www.youtube.com/watch?v=5gff3AGv7o8.
[21] Dataset: Valorant Youtube video. https://www.youtube.com/watch?v=dqhVl11DypA.
[22] Deploying NVIDIA Triton at Scale with MIG and Kubernetes. https://bit.ly/31X8zVQ.
[23] FFmpeg Official Website. https://www.ffmpeg.org/.
[24] gRPC Github Repository. https://github.com/grpc/grpc.
[25] H.264 Official Website . https://www.itu.int/rec/T-REC-H.264-200305-S/en.
[26] H.265 Official Website . https://www.itu.int/rec/T-REC-H.265.
[27] How Long is Twitch Stream Delay. https://onetwostream.com/blog/twitch-delay/.
[28] How to ensure a full live-streaming experience. https://www.techradar.com/news/how-to-ensure-a-full-live-streaming-experience.
[29] Intel® Core™ i9-9900K Processor Specifications. https://ark.intel.com/content/www/us/en/ark/products/186605/intel-core-i99900k-processor-16m-cache-up-to-5-00-ghz.html.
[30] Kakadu Official Website. https://kakadusoftware.com/.
[31] Kubernetes Load Balancer Document. https://kubernetes.io/docs/concepts/services-networking/_print/.
[32] libjpeg-turbo Github Repository. https://github.com/libjpeg-turbo/libjpeg-turbo.
[33] libvpx Github Repository. https://github.com/webmproject/libvpx.
[34] Most watched games on Twitch in 2020. https://sullygnome.com/games/2020/watched.
[35] NGCodec uses Amazon EC2 F1 instances with custom FPGAs running 4k Video Compression. www.prlog.org/12604725.
[36] NGINX Load Balancer Document. https://docs.nginx.com/nginx/admin-guide/load-balancer/http-load-balancer/.
[37] NVIDIA A100 Official Website. https://www.nvidia.com/en-us/data-center/a100/.
[38] NVIDIA Maxine Hits the Scene to Create Real-Time Video Experiences. https://blogs.nvidia.com/blog/2021/04/12/ai-real-time-video-maxine/.
[39] NVIDIA Maxine Official Website. https://developer.nvidia.com/maxine.
[40] NVIDIA NVENC Documentation. https://docs.nvidia.com/video-technologies/video-codec-sdk/ffmpeg-with-nvidia-gpu/.
[41] NVIDIA T4 Official Website. https://www.nvidia.com/en-gb/data-center/tesla-t4/.
[42] NVIDIA TensorRT Official Website. https://developer.nvidia.com/tensorrt.
[43] NVIDIA V100 Official Website. https://www.nvidia.com/en-gb/data-center/tesla-v100/.
[44] PNG Standard Website. https://www.iso.org/standard/29581.html.
[45] PyTorch Official Website. https://pytorch.org/.
[46] Real Time Streaming Protocol (RTSP) Specification. https://tools.ietf.org/html/rfc2326.
[47] Real-time Transport Protocol (RTP) Specification. https://tools.ietf.org/html/rfc3550.
[48] Scalable Video Coding (SVC) Extension for WebRTC. https://www.w3.org/TR/webrtc-svc/.
[49] Snapdragon 855+/860 Mobile Platform. https://www.qualcomm.com/products/snapdragon-855-plus-and-860-mobile-platform.
[50] SoftBank Solves Key Mobile Edge Computing Challenges Using NVIDIA Maxine. https://bit.ly/3dJNq47.
[51] Streaming Ad Revenue to Double by 2026. https://yhoo.it/329mAjP.
[52] Sub-Second Latency Streaming & Live Viewer Interactivity Changing the Video Landscape. https://www.limelight.com/resources/tech-brief/sub-second-latency-streaming-changing-the-video-landscape/.
[53] TensorRT Official Website. https://developer.nvidia.com/tensorrt.
[54] Touchcast Unveils World's First AI-Powered Event Platform Boosted by NVIDIA Maxine Technology. https://touchcast.com/newsroom/nvidia-partnership?utm_campaign=NVIDIA%20Partnership&utm_source=NVIDIA.
[55] Toward A Practical Perceptual Video Quality Metric. https://netflixtechblog.com/toward-a-practical-perceptual-video-quality-metric-653f208b9652.
[56] Twitch Broadcasting Guidelines. https://stream.twitch.tv/encoding/.
[57] Twitch Official Website. https://www.twitch.tv/.
[58] Twitch Revenue and Usage Statistics (2022). https://www.businessofapps.com/data/twitch-statistics/.
[59] Twitch Statistics. https://backlinko.com/twitch-users.
[60] Twitch Stream Delay: Everything You Should Know. https://bit.ly/3HGbPpI.
[61] VP9 codec parameters. https://www.webmproject.org/docs/encoder-parameters/.
[62] WebM Official Website. https://www.webmproject.org/.
[63] WebRTC 1.0: Real-Time Communication Between Browsers. https://www.w3.org/TR/webrtc/.
[64] WebRTC Official Website. https://webrtc.org/.
[65] WHAT AUDIENCES EXPECT FROM LIVE VIDEO. https://lp.livestream.com/rs/582-GOU-684/images/NYmag.pdf.
[66] Wowza Transcoding Documents. https://www.wowza.com/docs/wowza-transcoder.
[67] Xiaomi Mi9 Specifications. https://www.gsmarena.com/xiaomi_mi_9-9507.php.
[68] YouTube dataset (Product review). https://www.youtube.com/watch?v=qY2BdjhPmSE.
[69] Youtube Live Encoding Guidelines. https://support.google.com/youtube/answer/2853702?hl=en.
[70] Youtube Website. https://www.twitch.tv/.
[71] Duin Baek, Mallesham Dasari, Samir R Das, and Jihoon Ryoo. 2021. dcSR: Practical Video Quality Enhancement Using Data-Centric Super Resolution. (2021).
[72] Duin Baek, Mallesham Dasari, Samir R. Das, and Jihoon Ryoo. 2021. DcSR: Practical Video Quality Enhancement Using Data-Centric Super Resolution. In *Proceedings of the 17th International Conference on Emerging Networking EXperiments and Technologies (CoNEXT '21)*. Association for Computing Machinery, New York, NY, USA, 336–343. https://doi.org/10.1145/3485983.3494856
[73] Gisle Bjontegaard. 2001. Calculation of average PSNR differences between RD-curves. *VCEG-M33* (2001).
[74] Ying Chen, Qing Li, Aoyang Zhang, Longhao Zou, Yong Jiang, Zhimin Xu, Junlin Li, and Zhenhui Yuan. 2021. Higher quality live streaming under lower uplink bandwidth: an approach of super-resolution based video coding. In *Proceedings of the 31st ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*. 74–81.
[75] Daniel Crankshaw, Xin Wang, Guilio Zhou, Michael J Franklin, Joseph E Gonzalez, and Ion Stoica. 2017. Clipper: A {Low-Latency} Online Prediction Serving System. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. 613–627.
[76] M. Dasari, A. Bhattacharya, S. Vargas, P. Sahu, A. Balasubramanian, and S. R. Das. 2020. Streaming 360○ Videos using Super-resolution. In *Proceedings of the IEEE International Conference on Computer Communications (INFOCOM)*.
[77] Florin Dobrian, Vyas Sekar, Asad Awan, Ion Stoica, Dilip Joseph, Aditya Ganjam, Jibin Zhan, and Hui Zhang. 2011. Understanding the impact of video quality on

user engagement. *ACM SIGCOMM computer communication review* 41, 4 (2011), 362–373.

[78] Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. 2015. Image super-resolution using deep convolutional networks. *IEEE transactions on pattern analysis and machine intelligence* 38, 2 (2015), 295–307.

[79] Sadjad Fouladi, John Emmons, Emre Orbay, Catherine Wu, Riad S Wahby, and Keith Winstein. 2018. Salsify: Low-latency network video through tighter integration between a video codec and a transport protocol. In *15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18)*. 267–282.

[80] David Freedman, Robert Pisani, and Roger Purves. 2007. Statistics (international student edition). *Pisani, R. Purves, 4th edn. WW Norton & Company, New York* (2007).

[81] Arpan Gujarati, Reza Karimi, Safya Alzayat, Wei Hao, Antoine Kaufmann, Ymir Vigfusson, and Jonathan Mace. 2020. Serving {DNNs} like Clockwork: Performance Predictability from the Bottom Up. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. 443–462.

[82] Qi Huang, Petchean Ang, Peter Knowles, Tomasz Nykiel, Iaroslav Tverdokhlib, Amit Yajurvedi, Paul Dapolito IV, Xifan Yan, Maxim Bykov, Chuen Liang, et al. 2017. SVE: Distributed video processing at Facebook scale. In *Proceedings of the 26th Symposium on Operating Systems Principles*. 87–103.

[83] Junchen Jiang, Shijie Sun, Vyas Sekar, and Hui Zhang. 2017. Pytheas: Enabling data-driven quality of experience optimization using group-based exploration-exploitation. In *14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17)*. 393–406.

[84] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. 2016. Perceptual losses for real-time style transfer and super-resolution. In *European conference on computer vision*. Springer, 694–711.

[85] Jaehong Kim, Youngmok Jung, Hyunho Yeo, Juncheol Ye, and Dongsu Han. 2020. 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*. 107–125.

[86] Jiwon Kim, Jung Kwon Lee, and Kyoung Mu Lee. 2016. Accurate image super-resolution using very deep convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1646–1654.

[87] X. Kong, H. Zhao, Y. Qiao, and C. Dong. 2021. ClassSR: A General Framework to Accelerate Super-Resolution Networks by Data Characteristic. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, Los Alamitos, CA, USA, 12011–12020. https://doi.org/10.1109/CVPR46437.2021.01184

[88] Royson Lee, Stylianos I Venieris, Lukasz Dudziak, Sourav Bhattacharya, and Nicholas D Lane. 2019. Mobisr: Efficient on-device super-resolution through heterogeneous mobile processors. In *The 25th Annual International Conference on Mobile Computing and Networking*. 1–16.

[89] Bee Lim, Sanghyun Son, Heewon Kim, Seungjun Nah, and Kyoung Mu Lee. 2017. Enhanced deep residual networks for single image super-resolution. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*. 136–144.

[90] Hongqiang Harry Liu, Ye Wang, Yang Richard Yang, Hao Wang, and Chen Tian. 2012. Optimizing cost and performance for content multihoming. In *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*. 371–382.

[91] Zhenxiao Luo, Zelong Wang, Jinyu Chen, Miao Hu, Yipeng Zhou, Tom ZJ Fu, and Di Wu. 2021. CrowdSR: enabling high-quality video ingest in crowdsourced livecast via super-resolution. In *Proceedings of the 31st ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*. 90–97.

[92] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. 2017. Neural adaptive video streaming with pensieve. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. 197–210.

[93] Devdeep Ray, Jack Kosaian, KV Rashmi, and Srinivasan Seshan. 2019. Vantage: optimizing video upload for time-shifted viewing of social live streams. In *Proceedings of the ACM Special Interest Group on Data Communication*. 380–393.

[94] Francisco Romero, Qian Li, Neeraja J Yadwadkar, and Christos Kozyrakis. 2021. {INFaaS}: Automated Model-less Inference Serving. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*. 397–411.

[95] Haichen Shen, Lequn Chen, Yuchen Jin, Liangyu Zhao, Bingyu Kong, Matthai Philipose, Arvind Krishnamurthy, and Ravi Sundaram. 2019. Nexus: A GPU cluster engine for accelerating DNN-based video analysis. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*. 322–337.

[96] 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.

[97] Chuanqi Tan, Fuchun Sun, Tao Kong, Wenchang Zhang, Chao Yang, and Chunfang Liu. 2018. A survey on deep transfer learning. In *International conference on artificial neural networks*. Springer, 270–279.

[98] Yiding Wang, Weiyan Wang, Duowen Liu, Xin Jin, Junchen Jiang, and Kai Chen. 2021. Enabling Edge-Cloud Video Analytics for Robotics Applications. In *Proceedings of the IEEE International Conference on Computer Communications, Virtual Conference*. 10–13.

[99] Yiding Wang, Weiyan Wang, Duowen Liu, Xin Jin, Junchen Jiang, and Kai Chen. 2021. Enabling Edge-Cloud Video Analytics for Robotics Applications. In *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*. 1–10. https://doi.org/10.1109/INFOCOM42981.2021.9488801

[100] Yiding Wang, Weiyan Wang, Junxue Zhang, Junchen Jiang, and Kai Chen. 2019. Bridging the Edge-Cloud Barrier for Real-Time Advanced Vision Analytics. In *Proceedings of the 11th USENIX Conference on Hot Topics in Cloud Computing (HotCloud'19)*. USENIX Association, USA, 18.

[101] 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*. 1–14.

[102] 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.

[103] Xiaoqi Yin, Abhishek Jindal, Vyas Sekar, and Bruno Sinopoli. 2015. A control-theoretic approach for dynamic adaptive video streaming over HTTP. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*. 325–338.

[104] Anlan Zhang, Chendong Wang, Bo Han, and Feng Qian. 2017. YuZu: Super-resolution Enhanced Volumetric Video Streaming. In *14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 22)*. 393–406.

[105] Anlan Zhang, Chendong Wang, Bo Han, and Feng Qian. 2021. Efficient Volumetric Video Streaming Through Super Resolution. In *Proceedings of the 22nd International Workshop on Mobile Computing Systems and Applications*. 106–111.

[106] Cong Zhang and Jiangchuan Liu. 2015. On Crowdsourced Interactive Live Streaming: A Twitch.Tv-Based Measurement Study. In *Proceedings of the 25th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV '15)*. 55–60.

[107] Yulun Zhang, Yapeng Tian, Yu Kong, Bineng Zhong, and Yun Fu. 2018. Residual dense network for image super-resolution. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2472–2481.

[108] Zhengdong Zhang and Vivienne Sze. 2017. FAST: A framework to accelerate super-resolution processing on compressed videos. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 19–28.

[109] Lianmin Zheng, Chengfan Jia, Minmin Sun, Zhao Wu, Cody Hao Yu, Ameer Haj-Ali, Yida Wang, Jun Yang, Danyang Zhuo, Koushik Sen, et al. 2020. Ansor: Generating high-performance tensor programs for deep learning. In *14th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 20)*. 863–879.

[110] Size Zheng, Yun Liang, Shuo Wang, Renze Chen, and Kaiwen Sheng. 2020. Flextensor: An automatic schedule exploration and optimization framework for tensor computation on heterogeneous system. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*. 859–873.

[111] Anfu Zhou, Huanhuan Zhang, Guangyuan Su, Leilei Wu, Ruoxuan Ma, Zhen Meng, Xinyu Zhang, Xiufeng Xie, Huadong Ma, and Xiaojiang Chen. 2019. Learning to coordinate video codec with transport protocol for mobile video telephony. In *The 25th Annual International Conference on Mobile Computing and Networking*. 1–16.

[112] Xiao Zhu, Subhabrata Sen, and Z Morley Mao. 2021. Livelyzer: analyzing the first-Mile ingest performance of live video streaming. In *Proceedings of the 12th ACM Multimedia Systems Conference*. 36–50.

Appendices are supporting material that has not been peer-reviewed.

A DESIGN DETAILS

Estimating anchor gain. Algorithm 1 shows how NeuroScaler estimates the gains of anchor frames.

Algorithm 1 Per-group Anchor Gain Estimation

```
    • candidates: frames within a group
 1: function EstimateGain(candidates, frames)
 2:   residuals = CalcResidual(frames)
 3:   N = |candidates|
 4:   while N > 0 do N -= 1
 5:     gain_max = −∞
 6:     for i = 1 to |candidates| do
 7:       if not candidates[index].done then
 8:         gain = CalcDiffResidual(residuals, i)
 9:         if gain > gain_max then
10:           index_max = i
11:           gain_max = gain
12:     candidates[index_max].done = true
13:     candidates[index_max].gain = gain_max
14:     UpdateResidual(residuals, index_max)
15: function CalcResidual(frames)
16:   residual = 0
17:   for i = 1 to |frames| do
18:     if frames[i]==KEY then residual = 0
19:     else residual += frames[i].residual
20:     frames[i].acc_residual = residual
21: function UpdateResidual(residuals, index)
22:   Δ = residuals[index]
23:   for i = index to |residuals| do
24:     if residuals[i] == 0 then break
25:     else residuals[i] -= Δ
```

Memory management. The NeuroScaler's memory manager manages device and host memory pools during the inference as follows:
Device memory: The memory manager pre-allocates the entire accelerator memory and equally divides it into N_1 number of fragments. Whenever NeuroScaler allocates/releases a super-resolution DNN, the memory manager reserves/frees the fragment. We use the number of fragments as $N_1 = 2$ because this is enough for hiding the device memory allocation latency; as a single super-resolution DNN fully utilizes an accelerator, we do not run multiple DNNs concurrently.
Host memory: The memory manager pre-allocates pinned host memory that is sized to have N_2 number of frames per resolution (240p, …, 2160p). Then, the memory manager divides the allocated memory into per-resolution fragments. Whenever NeuroScaler allocates/releases a video frame, the memory manager reserves/frees the fragment. We set the initial number of per-resolution fragments as $N_2 = 40$ and double the number when there are no available fragments.

B EXPERIMENTAL SETTINGS

Computing instances. Table 1 shows the specifications of AWS EC2 instances used in our evaluation. The price is calculated based on 3-year reserved instances in the US East(N. Virginia) region.

Instance type	GPUs	vCPUs	Mem	Price
g4dn.xlarge	1	4	16 GB	$0.227/hour
g4dn.2xlarge	1	8	32 GB	$0.325/hour
g4dn.4xlarge	1	16	64 GB	$0.520/hour
g4dn.8xlarge	1	32	128 GB	$0.940/hour
g4dn.16xlarge	1	64	256 GB	$1.880/hour
g4dn.12xlarge	4	48	192 GB	$1.690/hour
g5dn.2xlarge	1	8	16 GB	$0.524/hour
c6i.8xlarge	0	32	64 GB	$0.599/hour

Table 1: AWS EC2 instance specfications

VP9 parameters. We tune the VP9 encoding parameters to improve the quality gains of anchor frames. The default CBR (constant bitrate) mode does not include alternative reference frames, which provide high efficiency as anchor frames. Instead, we use the constrained VBR mode and turn on the alternative reference frame feature. Using this setting, we encode video by FFmpeg (v4.4) [23] as follows:

- NeuroScaler: `ffmpeg -i {input video} -c v libvpx-vp9 -vf scale={width}x{height} -minrate {bitrate*0.5} -maxrate {bitrate*1.5} -b v {bitrate} -keyint_min 120 -g 120 -auto-alt-ref 1 -lag-in-frames 16 -qmin 4 -qmax 48 -error-resilient 1 -quality realtime -speed {speed} -row-mt 1 -frame-parallel 1 {output video}`
- Default CBR: `ffmpeg -i {input video} -c v libvpx-vp9 -vf scale={width}x{height} -b v {bitrate} -keyint_min 120 -g 120 -qmin 4 -qmax 48 -error-resilient 1 -quality realtime -speed {speed} -row-mt 1 -frame-parallel 1 {output video}`

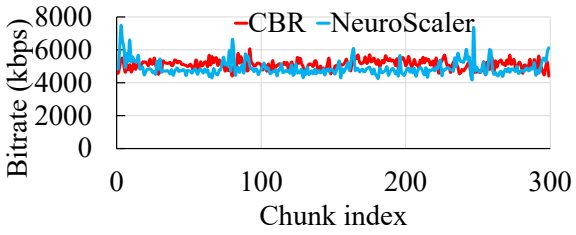


Figure 28: Bitrate comparison

To demonstrate the feasibility of the new configuration for live streaming, we analyzed the bitrate and the latency:
Bitrate: Figure 28 illustrates per-chunk bitrate using a 720p, 60 fps video [19] in our dataset. The target bitrate is set to 4125 kbps. The result shows that 1) our setting shows a similar bitrate compared to the default CBR mode, and 2) the average bitrate of our setting (4888 kbps) is closer to the target compared to that of the CBR version (5104 kbps).
Latency: Using alternative reference frames incurs additional latency because the VP9 codec needs to look up candidate frames in the future. The delay can be adjusted by `-lag-in-frame` option, which is the maximum number of frames into the future that the codec can refer [61]. We set it as 16, which causes 266 ms delay, unless otherwise noted.
JPEG/JPEG2000 parameters. Table 2 shows the quantization parameters (QP) for encoding JPEG/JPEG200 images; a higher QP value results in larger files (with higher quality). We adjust the

Fraction of anchor frames	JPEG	JPEG2000
10% < Fraction ≤ 15%	-	85
7.5% < Fraction ≤ 10%	80	90
5% < Fraction ≤ 7.5%	90	95
Fraction ≤ 5%	95	95

Table 2: JPEG/JPEG2000 QP values

QP values according to the fraction of anchor frames to maximize video quality while guaranteeing bitrate smaller than that of the per-frame encoding (§8). When the fraction of anchor frames becomes higher than 10% and 15%, JPEG and JPEG2000 cannot meet the bitrate constraint, respectively.

Content	Method	(#blocks, #channels)	Fraction of anchor frames
Chat	Per-frame	8, 24	-
	Selective	8, 32	25%
GTA	Per-frame	8, 20	-
	Selective	8, 32	15%
LoL	Per-frame	8, 20	-
	Selective	8, 32	20%
Fortnite	Per-frame	8, 10	-
	Selective	8, 32	15%
Valorant	Per-frame	8, 20	-
	Selective	8, 32	15%
Minecraft	Per-frame	8, 10	-
	Selective	8, 32	15%

Table 3: Baseline configurations

Baseline configuration. Table 3 illustrates the configurations of the per-frame and selective baseline that achieve a similar quality as NeuroScaler (average ΔPSNR=0.215 dB). We set #blocks, #channels, and the fraction of anchor frames as 8, 32, and 7.5% for NeuroScaler, respectively. The baseline configurations vary across contents because each has a different amount of scene complexity and temporal redundancy, which affects the gains of a super-resolution DNN and selective inference, respectively.

C ADDITIONAL RESULTS

	Instance type	Numbers (per 100 streams)
Per-frame (SW)	g4dn.12xlarge	100
Per-frame (HW)	g4dn.12xlarge	100
Selective (SW)	g4dn.12xlarge	50
Selective (HW)	g4dn.xlarge	100
NeuroScaler	g4dn.xlarge	34

Table 4: Cost-effective settings

Original	Per-frame SR	Key+Uniform SR	NeuroScaler SR
34.27	86.42	85.71	86.57

Table 5: Video quality in VMAF (Content: League of Legends [19])

CPU specification	#threads	Throughput (fps)
Intel i9-9900K (3.6 GHz)	1	89.4
	2	140.0
	4	184.992

Table 6: Decoding throughput (Desktop-class CPU)

Method	Inference	Encoding
Per-frame (SW)	4 GPU	16 vCPU
Per-frame (HW)	4 GPU	1 GPU
Selective (SW)	0.92 GPU	16 vCPU
Selective (HW)	0.92 GPU	1 GPU
NeuroScaler	0.33 GPU	0.25 vCPU

Table 7: Resource usage per stream (A single vCPU can run a single CPU thread [14].)

Process	Cost-effective	Latency-sensitive
Instance	`g4dn.xlarge`	`g5.2xlarge`
Decode	10.0ms (± 4.95ms)	5.61ms (± 4.04ms)
Schedule	0.155ms (± 0.024ms)	3.57ms (± 5.51ms)
Infer	106ms (± 26.5ms)	41.5ms (± 8.41ms)
Encode	18.0ms (± 0.790ms)	12.6ms (± 0.32ms)
Queue	557ms (± 305ms)	27.4ms (± 20.8ms)
E2E	669ms (± 338ms)	90.8ms (± 25.8ms)

Table 8: NeuroScaler's latency analysis (average ± standard deviation)

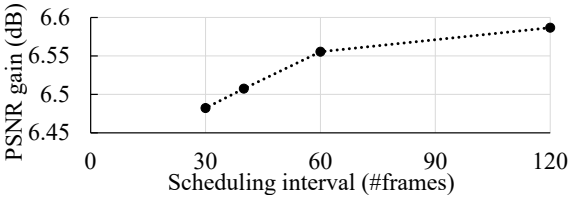


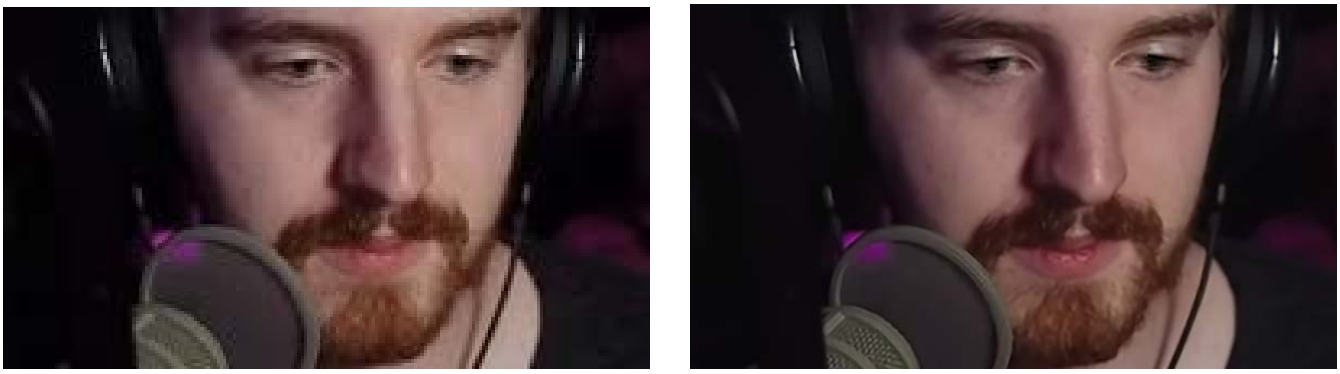
Figure 29: Increasing the scheduling interval results in higher quality gain. The x-axis represents the scheduling interval in # frames. The experiment was conducted with a video [16], which GOP is 120. 10% of frames were selected as anchor frames.

(a) Original (720p) / PSNR: 32.3 dB (b) NeuroScaler (2160p) / PSNR: 39.9 dB

Figure 30: League of Legends video snapshot (Source: Youtube [19])

(a) Original (720p) / PSNR: 33.2 dB (b) NeuroScaler (2160p) / PSNR: 38.3 dB

Figure 31: Valorant video snapshot (Source: Youtube [21])



(a) Original (720p) / PSNR: 35.2 dB (b) NeuroScaler (2160p) / PSNR: 42.8 dB

Figure 32: Just Chatting video snapshot (Source: Youtube [16])

D ARTIFACT APPENDIX

Abstract

NeuroScaler's artifact contains the source code, scripts, and documentation, which is hosted as a Github repository. The repository provides detailed instructions to build, run, and evaluate NeuroScaler.

Scope

The artifact allows to validate the end-to-end processing improvement and quality gain, shown in Figure 13(a) and 13(b), respectively. One can use the artifact for their own DNN and video or extend the artifact to integrate with their video streaming applications.

Contents

The artifact provides the source code, scripts, and documentation to run NeuroScaler. It provides how to NeuroScaler in three steps: 1) generate datasets (e.g., videos, DNNs), 2) measure the end-to-end throughput, and 3) measure the video quality.

Hosting

NeuroScaler is available on Github, which URL is https://github.com/kaist-ina/neuroscaler-public. The artifact is provided in the main branch of the repository.

Requirements

The following sofware/hardware are required to run NeuroScaler:

- Operating system: Ubuntu 16.04 or higher versions
- Software: Python 3.6 or higher versions (Required package: xlrd), Docker, NVIDIA Docker
- Hardware: NVIDIA GPUs