NEMO: Enabling Neural-enhanced Video Streaming on Commodity Mobile Devices

Hyunho Yeo Chan Ju Chong Youngmok Jung Juncheol Ye Dongsu Han
KAIST

ABSTRACT

The demand for mobile video streaming has experienced tremendous growth over the last decade. However, existing methods of video delivery fall short of delivering high-quality video. Recent advances in neural super-resolution have opened up the possibility of enhancing video quality by leveraging client-side computation. Unfortunately, mobile devices cannot benefit from this because it is too expensive in computation and power-hungry.

To overcome the limitation, we present NEMO, a system that enables real-time video super-resolution on mobile devices. NEMO applies neural super-resolution to a few select frames and transfers the outputs to benefit the remaining frames. The frames to which super-resolution is applied are carefully chosen to maximize the overall quality gains. NEMO leverages fine-grained dependencies using information from the video codec and strives to provide guarantees in the quality degradation compared to per-frame super-resolution. Our evaluation using a full system implementation on Android shows NEMO improves the overall processing throughput by x11.5, reduces energy consumption by 88.6%, and maintains device temperatures at acceptable levels compared to per-frame super-resolution, while ensuring high video quality. Overall, this leads to a 31.2% improvement in quality of experience for mobile users.

CCS CONCEPTS

• **Information systems** → **Multimedia streaming**; • **Computing methodologies** → *Computer vision*; • **Human-centered computing** → *Ubiquitous and mobile computing*.

KEYWORDS

Video streaming, video codec, mobile computing, super-resolution, deep neural networks

ACM Reference Format:
Hyunho Yeo, Chan Ju Chong, Youngmok Jung, Juncheol Ye, and Dongsu Han. 2020. NEMO: Enabling Neural-enhanced Video Streaming on Commodity Mobile Devices. In *The 26th Annual International Conference on Mobile Computing and Networking (MobiCom '20), September 21–25, 2020, London, United Kingdom.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3372224.3419185

MobiCom '20, September 21–25, 2020, London, United Kingdom

ACM ISBN 978-1-4503-7085-1/20/09…$15.00
https://doi.org/10.1145/3372224.3419185

1 INTRODUCTION

Mobile video streaming has experienced substantial growth over the last decade. A recent market report indicates the demand for mobile video has quadrupled over the last seven years [26] and smartphones/tablets account for 62% of viewership [11]. On YouTube alone, more than 70% of all video consumption happened via mobile devices [10]. At the same time, due to users' steep expectations for quality, delivering a high quality of experience (QoE) [36] to mobile users has become of paramount importance. To meet the seemingly insatiable demand, both mobile carriers and CDNs have made onerous efforts to scale bandwidth [1–4, 29]. Efforts to maximize user QoE given the bandwidth constraint have also made significant advances in adaptive streaming [37, 48, 51, 53, 63]. However, the fundamental limitation is that user QoE heavily depends on the bandwidth.

Recent advances in neural-enhanced video streaming [40, 44, 60, 61] create a new opportunity for QoE enhancement using client-side computation independent of the network bandwidth. In particular, NAS [61] applies neural super-resolution on video frames to produce high quality video. Although this approach brings significant benefits, it relies critically on the client's computational capacity, which mobile devices cannot spare. Compared to desktop-class GPUs used in prior works [44, 60, 61], mobile devices inherently have meek computational capacities coupled with strenuous power constraints. Even the state-of-the-art mobile super-resolution [54], targeted for images, falls wildly short of supporting real-time video playback. Even with one of the latest Qualcomm chips (Snapdragon 855), real-time video super-resolution to 1080p is not feasible, producing impractical delays, unsustainable rates of battery drain, and large amounts of heat dissipation [49] that causes discomfort to users, as we demonstrate in §3.

Motivated by these limitations, we present NEMO, a new video delivery framework that enables real-time video super-resolution on mobile devices. NEMO exploits temporal redundancies within a video to drastically reduce computation. NEMO applies super-resolution to a few select frames but reuses the result of super-resolution to benefit the entire video. The resulting system is one that effectively reduces computation (i.e., number of operations) by a factor of 9.5-51.1 (19.5 on average). It offers high-quality video even when network conditions are not favorable, while maintaining battery consumption and device temperatures at acceptable levels.

NEMO consists of two novel components. First, at the server-side, the media server analyzes a video to select frames to apply super-resolution for maximizing the quality gains. This information is captured in what we call a *cache profile*, provided to clients alongside the video content at the beginning of each streaming session. Second, at the client-side, by referring to the *cache profile*, the mobile client applies neural super-resolution to the select frames, referred to as *anchor points*. Anchor points that have been

up-scaled using super-resolution are cached so they can be reused for up-scaling non-anchor frames.

Creating a full-fledged system that achieves video super-resolution at real-time on mobile devices, however, involves solving a number of non-trivial challenges:

- First, to transfer the benefit of neural super-resolution of select frames to the entire video, we leverage fine-grained information from a video codec regarding frame dependencies. However, this must be done carefully to maximize the computational savings without drastically compromising the quality.
- Second, anchor point selection greatly affects the video quality. However, the number of possible sets of anchor points are on the order of $2^{|frame|}$, making exhaustive search infeasible.
- Finally, mobile devices are heteregenuous with varying computing capacities. The system must be able to adapt to various mobile devices to enable real-time video super-resolution on devices ranging from high-end phones to cheaper entry-level phones.

NEMO addresses these challenges by introducing new system designs. To effectively transfer the gain of neural super-resolution to successive frames, we leverage fine-grained frame dependencies processed in a video codec [17, 18, 28]. To efficiently enable this, we develop an *SR-integrated codec* that incorporates both a cache and a super-resolution mechanism into an existing video codec. The codec internally applies neural super-resolution to the anchor points, and uses frame dependencies and cached high-resolution frames to up-scale the remaining frames. Next, to select anchor points, we develop an efficient approximation algorithm that reduces the search space from $2^{|frame|}$ (all possible sets) to $|frame|$. The algorithm allows us to choose a minimal number of anchor points that ensures the quality loss due to reduced computation is within a given threshold (e.g., 0.5 dB in peak-signal-to-nose-ratio). However, as the temporal redundancy of a video widely vary across chunks within a video, a few outlier chunks may exhibit extremely low redundancy (e.g., scene transitions, fastforward, scene cut), requiring a disproportionately high number of anchor points. To guarantee real-time processing for every chunk, we set an upper-bound on the number of anchor points and choose a quality margin of 0.5 dB which is satisfied for the majority of chunks; the upper bound minimally degrades the average video quality (0.03 dB). Finally, mobile devices have different computing capacities, and the number of anchor points required differs across videos. To enable real-time processing under device- and video-specific constraints, the server offers multiple DNN options and produces cache profiles corresponding to each option. The client then selects the combination that best suits its computational capacity.

We evaluate NEMO using a full system implementation using the state-of-the-art commercial video codec, VP9 [28], and standard Android video player, Exoplayer [14]. Our evaluation using two commercial smartphones and a tablet on 30 Youtube videos shows that NEMO improves the video processing throughput by x7.8-x21.5, reduces energy consumption by 88.2%-89.4%, and maintains device temperatures at a level that does not cause discomfort to the user, compared to per-frame super-resolution. At the same time, it limits the quality degradation to an average of 0.41 dB in Peak-Signal-to-Noise-Ratio (PSNR) compared to per-frame super-resolution, providing an average gain of 0.99-6.34 dB compared to traditional streaming. This delivers 20.1-49.9% QoE improvement (31.2% on average) for mobile clients that use adaptive streaming. NEMO's source code is available at [23].

In summary, we make three key contributions:

- **Neural video enhancement on mobile devices:** To the best of our knowledge, NEMO is the first to enable real-time video super-resolution on commodity mobile devices.
- **Cache framework for video super-resolution:** Our video codec integrates neural super-resolution and maximizes the benefit of reusing super-resolution results, delivering a x11.5 speed-up compared to per-frame super-resolution.
- **Resource optimization for video super-resolution:** NEMO minimizes the overhead of video super-resolution by selectively applying it to minimal number of anchor points that deliver the largest overall quality improvement.

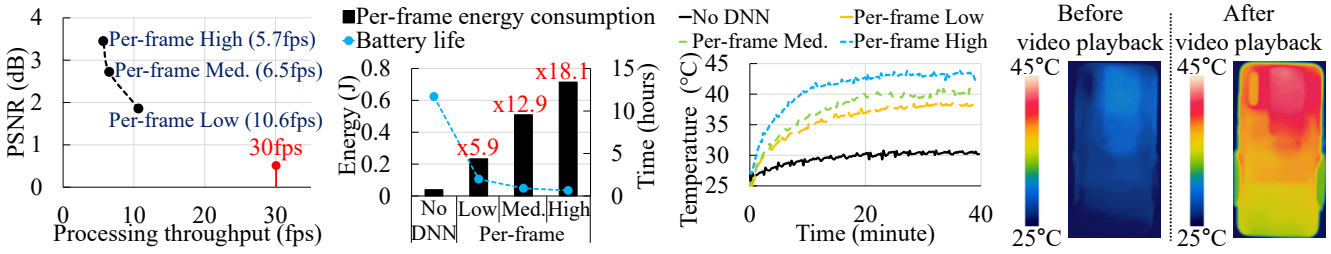
2 BACKGROUND

Adaptive streaming (e.g., DASH [22], HLS [6]) is capable of handling unpredictable bandwidth variations. In adaptive streaming, a server encodes a video at multiple bitrates and divides them into fixed length chunks, typically of 2-10 seconds duration. A client then uses an adaptive bitrate algorithm (ABR) to select a suitable video quality based on its current network bandwidth. Many algorithmic/systemic advances [43, 47, 55–58, 62] have been made to optimize bitrate and server selections. However, adaptive streaming still suffers from a fundamental limitation in that video quality heavily depends on the available network bandwidth.

Super-resolution recovers a high-resolution image from a lower-resolution version. Recent advances in convolutional neural networks (CNN) have led to significant performance improvements in super-resolution. A large body of work [37, 48, 51, 53, 63] has been devoted to make super-resolution faster and more accurate. Despite these efforts, it remains to be a computationally *expensive* task; this motivates our work for enabling it on mobile devices in a streaming context.

Content-aware neural adaptive streaming. NAS [61] applies super-resolution on top of adaptive streaming. In NAS, a server trains a *content-aware* super-resolution DNN for each video and provides the DNN alongside the video. A client then uses *integrated* ABR to determine whether to download the DNN for quality enhancement or a video chunk at a specific bitrate. After downloading the DNN, the client runs it on own computing device to watch a high-resolution video from a lower quality transmission. However, mobile devices cannot benefit from NAS because it requires powerful desktop-class GPUs (see §3).

Video codecs (e.g., H.26x [17, 18], VPx [28]) perform compression by 1) partitioning a frame into non-overlapping blocks and 2) applying either inter- or intra-frame coding to each block. For an inter-coded block, the codecs find the most similar reference block in the previous frames and encodes the offsets of the selected frame and block, which are called *reference index* and *motion vector*, respectively; the motion vector is commonly represented in quarter-pixel granularity [12, 16]. Next, the difference in pixel values between the the reference and the target block, referred to as



(a) Processing throughput (b) Energy consumption (c) Surface temperature (d) Infra-red thermal images

Figure 1: Motivating measurements on a recent smartphone

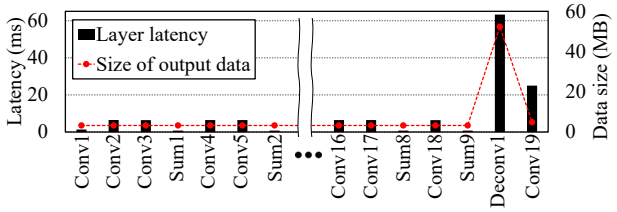


Figure 2: Per-layer latency benchmark

residual, is encoded. Intra-coded blocks follow the same process, except they select blocks within the same frame as reference. In this study, we leverage the compressed video information to accelerate video super-resolution.

3 MOTIVATION

To enable neural video enhancement in a streaming context, a client must apply super-resolution DNNs to each frame at a real-time throughput (i.e., 24-30 fps). However, super-resolution DNNs are far too computationally expensive for real-time computation on a mobile device. Furthermore, heavy computations directly impact the rate of battery consumption and the surface temperature of mobile devices, which greatly deteriorate user experiences [49]. To clearly illustrate these challenges, we measure the overhead of running neural super-resolution on a recent smartphone (Xiaomi Mi9 [31]) equipped with a Qualcomm Snapdragon 855 (Kryo 485 CPU, Adreno 640 GPU), one of the latest high-end mobile processors released in March, 2019.

Motivating measurements. We played a 240p video [34] using Google's Exoplayer [14] and apply super-resolution frame-by-frame on the mobile GPU to obtain a 960p video; here, we set the screen brightness to maximum. We used three DNNs of varying quality levels from NAS [61] (e.g., Low, Medium, High) whose computation requirements differ by up to a factor of 28. We measured the video processing throughput (i.e., frames per second), energy consumption, and surface temperature of the device while streaming a video for 40 continuous minutes and report their average.

Throughput & Quality. Figure 1(a) shows the plotting of video quality gain of per-frame super-resolution against the average video processing throughput. Super-resolution DNNs significantly improve the quality between 1.9 dB (Low) and 3.5 dB (High) on average. However, per-frame super-resolution only achieves 5.7-10.6 fps, falling widely short of real-time throughput. Further reducing the DNN size would result in a minimal quality gain due to the steep tradeoff between computation and quality.

Energy & Battery concerns. We measured the energy consumption of the smartphone using the Monsoon High Voltage Power Monitor [21].[1] Figure 1(b) shows the average per-frame energy consumption (bar graph) and the expected battery life for video playback (dotted line). The results show that running super-resolution (SR) DNNs requires x5.9-x18.1 energy consumption compared to video playback without DNN processing. As a result, the expected battery life decreases from 11.7 hours to 0.6-2.0 hours.

Temperature. Per-frame DNN inference also causes large amounts of heat dissipation, which rapidly increase the surface temperature of the smartphone. Figure 1(c) shows the surface temperature of the hottest point over time measured using the FLIR ONE thermographic camera [13]. Figure 1(d) shows thermographic images of the rear side of the smartphone before and after video streaming using the higest-quality DNN. The results indicate that the surface temperature quickly reaches 40.8°C and 43.9°C using the medium-quality and the high-quality DNN, respectively. Recent user experience studies [49, 52] indicate users feel discomfort at temperatures above 33-35°C and even pain at around 42-45°C. We observe that per-frame super-resolution can greatly damage user experiences.

Summary. From our experiment, we conclude that a naive application of super-resolution is impractical in the context of video streaming. To make neural video super-resolution practical on mobile phones, we must meet the real-time requirement while retaining the quality gain and reduce the computation to limit energy consumption and heat dissipation induced by super-resolution.

4 KEY DESIGN CHOICES

To overcome the limitations of per-frame super-resolution in §3, NEMO applies a super-resolution DNN only to a subset of frames and caches their outputs. We then reuse the outputs to up-scale the remaining frames, exploiting the temporal redundancy across frames. The goal is to amortize the computational overhead of a super-resolution DNN across many frames. At the same time, we want to provide a guarantee that the resulting video quality is within a small margin compared to the quality that per-frame super-resolution delivers. We ask ourselves a series of pivotal questions that lead to the key design choices we make in achieving the goal.

4.1 What to Cache?

Key observations. A DNN consists of multiple layers, and each layer delivers different levels of computation savings and quality

[1] Since recent smartphones are equipped with an integrated battery, we disassembled our device and replaced its battery with the Monsoon power monitor. The photos of our experiment setting are available at [23].

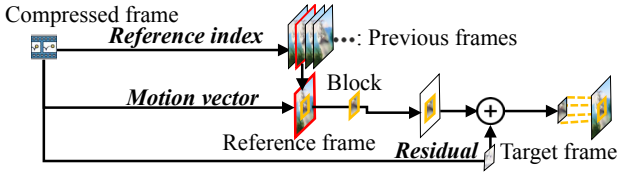


Figure 3: Frame dependencies processed in a codec [9]

loss by caching. Prior studies [38, 45, 59] that focus on object classification and detection cache the outputs of earlier convolution layers (also called feature maps) that contribute to the majority of computations for re-use. In contrast to object classification, super-resolution DNNs enlarge feature maps to reconstruct a high-resolution image and show drastically different computational characteristics. Figure 2 shows the per-layer latency of a super-resolution DNN (used in NAS [61]) measured using a recent smartphone [31]. It shows most of the latency occurs at the last couple of layers (Deconv1, Conv19). This means caching the outputs of intermediate convolution layers (Conv1-Conv18) would be ineffective in reducing computation.

Approach. NEMO applies neural super-resolution only to a subset of frames, referred to as *anchor points*. To make the most out of caching, we cache the final output (i.e., the high-resolution image). The remaining frames reuses the cached result to up-scale their resolutions. To maximize the benefit of reuse, we carefully transfer/reuse the pixels of cached frames using fine-grained frame dependencies (§4.2) and select an optimized set of anchor points that delivers the largest overall quality improvement (§4.3).

4.2 How to Reuse?

Key observations. To maximize the quality of frames up-scaled by cached high-resolution frames, we need fine-grained dependencies among frames to decide 1) which frame to reuse among previously reconstructed cached frames and 2) how to transfer each pixel of the cached frame to at which position within a target frame. However, as computing fine-grained frame dependencies among video frames is computationally expensive [39, 66, 67], running them on resource-constrained mobile devices can largely diminish the computation savings introduced by caching.

Fortunately, our key observation is that rich information about frame dependencies are embedded in a compressed video and already processed when a video is decoded. Figure 3 illustrates how frame dependency information within a codec is used to decode an inter-coded block. First, the codec uses *reference index* to extract a reference frame among previously decoded frames. Next, it applies *motion vector* to a source block in the reference frame to transfer the block to the target frame. Finally, it adds *residual* to the transferred block, which accounts for the difference between the predicted and original pixels.

Approach. To transfer the pixels of cached high-resolution frames to other frames, we leverage frame dependencies processed inside a video codec. To make this efficient, we develop an *SR-integrated codec* (§5.1) that internally applies neural super-resolution to the anchor points and uses frame dependencies and cached high-resolution frames to up-scale the remaining frames. In particular, it uses *reference index* and *motion vector* to transfer pixels of cached high-resolution frames to other frames and *residual* to compensate for the temporal difference during the transferal.

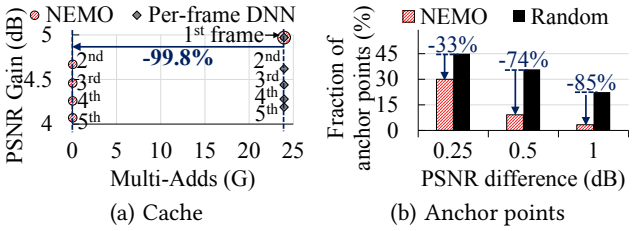


Figure 4: Potential benefits of NEMO

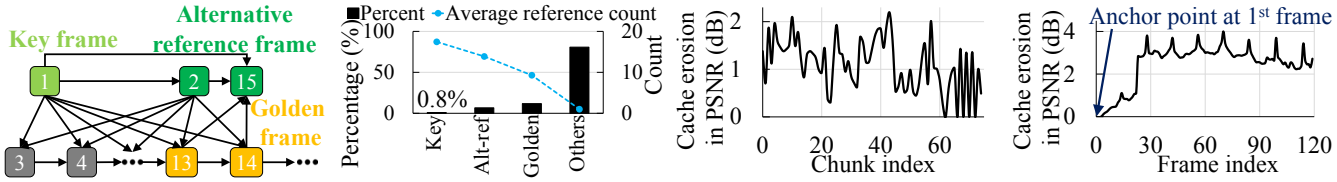
To demonstrate the potential benefits of the integrated codec, we measure the quality improvement in PSNR and the number of multiply-accumulate operations (i.e., $a \times b + c$); here, we apply the high-quality DNN from NAS [61] to a 240p video [34]. Figure 4(a) shows the result for five frames for NEMO and per-frame super-resolution. In NEMO, the first frame (i.e., anchor point) is processed using a super-resolution DNN, but the remaining non-anchor frames use the cached results, whereas per-frame super-resolution (SR) applies SR to every frame. We observe that both deliver similar quality improvements of 4.07 to 4.97 dB, but NEMO's caching cuts down multiply-accumulate operations by 99.8% for non-anchor frames (frames #2-#5) compared to per-frame SR. Using a recent smartphone [31], we observe that a non-anchor frame is processed very fast (≤15 ms), while an anchor point consumes more than 150 ms.

4.3 How to Guarantee Quality?

Key observations. We would like to ensure that the video quality NEMO delivers is within a small margin (e.g., $\leq 0.5dB$) compared to that of per-frame super-resolution. However, the quality improvement that each frame produces when selected as an anchor point widely varies between frames and largely depends on *frame dependency* and *cache erosion*. Thus, selecting an optimal set of anchor points is crucial.

Frame dependency: Compressed video frames have complex dependencies in the form of a directed acyclic graph. Figure 5(a) shows the dependency graph for 15 frames within a video [34] encoded with VP9; nodes and edges represent frames and dependencies, respectively. Figure 5(b) shows the percentage and the average reference count of each frame type. The VP9 codec has three special types of frames that show high degrees of reference: 1) key frames (0.8%) are the first frame of a group of picture (GOP); 2) alternative reference frames (6.8%) are non-visible frames solely used for improving inter-predictions of other frames; and 3) golden frames (11.8%) are frames that do not fall in to the two categories but are referenced multiple times. The remaining frames, which account for 80.6%, have no more than one dependent frame. Our key observation is that using frames with a high degree of reference as anchor points and reusing the result would deliver quality improvements to a larger number of frames.

Cache erosion: When a frame is up-scaled using cached high-resolution frames, quality degradation inevitably occurs. We refer to this as *cache erosion*. The degree of cache erosion is content-dependent. To better understand cache erosion, we measure it as the quality degradation compared to per-frame super-resolution when reusing cached results. For illustration, we use only key frames as anchor points. Figure 5(c) illustrates the average cache erosion over each



(a) Frame dependency (b) Frame type (c) Cache erosion per chunk (d) Cache erosion per frame

Figure 5: Key observations on anchor points

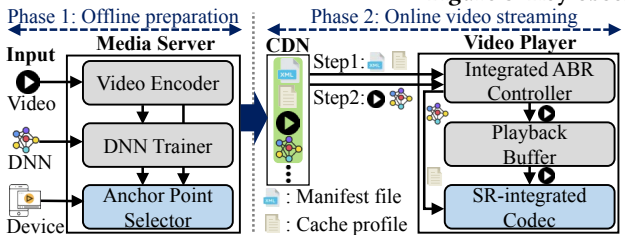


Figure 6: NEMO framework overview

(GOP-sized) video chunk within a video [34], where GOP is set to 120. The result indicates that cache erosion greatly varies across video chunks (between 0 to 2.19 dB in PSNR) due to varying levels of temporal redundancy. Figure 5(d) shows the frame-by-frame cache erosion of the 10th GOP in which scene changes frequently occur. The first frame is a key frame that is up-scaled using neural super-resolution, and the rest are up-scaled using cached high-resolution frames. As we move along the frames, erosion accumulates fast and soon reaches ~3 dB in PSNR at frame #25, largely negating the benefit of super-resolution. This indicates that selecting only key frames as anchor points does not ensure high quality and that anchor points must be carefully chosen to provide a guarantee on the resulting video quality.

Approach. NEMO aims to select an optimized set of anchor points tailored to each video that guarantees the quality loss is within a specified margin. For this, we design an efficient algorithm (§5.2) that estimates the resulting quality by a set of anchor points considering the impact of frame dependency and cache erosion in linear time ($O(|frame|)$). The algorithm allows us to choose a small number of anchor points while guaranteeing the quality.

Figure 4(b) shows the fraction of frames that are anchor frames within a video for NEMO and for a naive approach which randomly selects anchor points until the quality difference compared to per-frame super-resolution falls within {0.25, 0.5, 0.1}dB. NEMO reduces the number of anchor points by 33% to 85% compared to the random policy while providing the same quality level.

5 SYSTEM DESIGN

Figure 6 illustrates an overview of NEMO divided into offline and online phases.

Offline preparation. When a video is uploaded, a media server transcodes the video into multiple-bitrate versions and trains super-resolution DNNs using these videos as with content-aware adaptive streaming [61]. The server then selects a minimal set of anchor points that provides a guarantee that the average quality is within a specified margin from that of per-frame super-resolution (§5.2). The quality margin is a configurable parameter. However, each video requires a different number of anchor points, and each mobile device/processor also has different computing capacities. To support

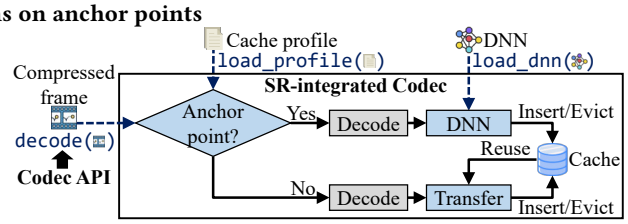


Figure 7: SR-integrated codec overview

real-time processing under these constraints, the server provides multiple performance options by providing DNNs of different size and quality. The server specifies each DNN and its corresponding cache profile in the manifest file along with a list of supported devices (§5.3).

Online video streaming. At the beginning of video streaming, a client's video player downloads a manifest file, as with standard adaptive streaming. NEMO's manifest file, however, also includes a set of available DNNs and their cache profiles. The client chooses a DNN and a cache profile based on its mobile processor type. It then downloads the cache profile, whose average size is around 0.3 KB per minute of video (i.e., one bit per frame). Next, as in content-aware adaptive streaming [61], the client runs integrated ABR to stream the selected DNN and the video simultaneously. The video player keeps the video chunk on the playback buffer and initializes the *SR-integrated codec* with the cache profile and the DNN. The codec then either applies the DNN or uses cached high-resolution frames to up-scale a frame by referring to the cache profile (§5.1). Finally, super-resoluted frames are rendered by the video player.

5.1 SR-Integrated Codec

Goal & Challenge. As compressed videos contain rich information about frame dependencies (§4.2), we aim to leverage this codec information to transfer super-resolution outputs over successive frames. However, typical codec implementations [17, 18, 28] provide limited decoding APIs that take a compressed frame and only return a decoded frame (raw pixel values) without any additional information.

Moreover, recent codecs (e.g., VP8/9 [28], AV1 [8]) have special frames, called alternative-reference frames, which are solely processed inside the codec to improve inter-prediction. These frames commonly have a high degree of reference (§4.3) and deliver large benefits as anchor points. Unfortunately, existing APIs do not allow access to these frames.

Approach. To leverage internal codec information, we develop an *SR-integrated codec* that incorporates both a cache mechanism and a super-resolution mechanism into an existing codec [28]. Figure 7 shows an overview of the SR-integrated codec that utilizes a cache profile and a deep neural network (DNN) for super-resolution. When a compressed frame is passed to the codec, it first checks

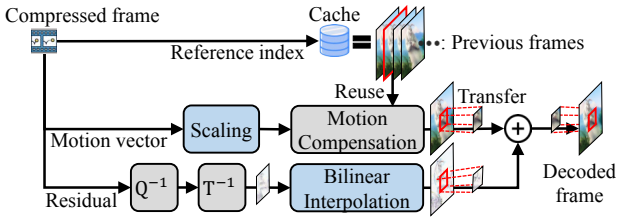


Figure 8: Reusing cached high-resolution frames [9] (Q^{-1}, T^{-1}: Inverse-quantization, -transform)

whether the frame is an anchor point or not by referring to the cache profile. If the frame is an anchor point, the codec up-scales it using the super-resolution DNN and caches the output. Otherwise, the codec uses frame dependency information and cached outputs to up-scale the current frame, which is also cached for later use.

The SR-integrated codec abstracts both neural and cache-based super-resolution and provides the same decoding API to that of the existing video codecs:

decode(`compressed frame`) → `super-resoluted frame`

Developers do not need to understand the internal mechanisms, enabling the codec to be easily deployed to applications with minimal development effort. We modify 302 lines of code to deploy our new codec on Android standard video player [14] (see Table 1).

Up-scaling using cached frames. We now explain how the SR-integrated codec up-scales non-anchor frames. A VP9 frame consists of non-overlapping blocks of *inter-coded blocks* or *intra-coded blocks* and is decoded on a per-block basis (§2).

Inter-coded blocks: Figure 8 illustrates the process of reusing cached high-resolution frames for inter-coded blocks. Inter-coded blocks contain reference indexes, motion vectors, and residual blocks. First, the codec uses a reference index to select a (high-resolution) reference among previously reconstructed frames in cache. Next, the codec scales the motion vector. For example, when it up-scales a frame from 240p to 960p, the motion vector is multiplied by 4. Motion compensation is applied to account for quarter-pixel motion. Using the motion vector, the codec transfers the target block from the reference frame to the current frame. Finally, the codec decodes and up-scales the residual block using lightweight bilinear interpolation and accumulates it to the transferred block to output a high-resolution block. To accelerate this, we apply a suite of NEON vectorized instructions [7] available on mobile CPUs. However, since bilinear up-sampling results in loss of high-frequency components, cache erosion is inevitably produced. In §5.2, we present an algorithm that effectively controls cache erosion by judiciously distributing anchor points.

Intra-coded blocks: Intra-coded blocks are predicted using nearby pixels within the same frame. This leads to the challenge when there is no reusable data in cache. Thus, we can use either bilinear interpolation or a super-resolution DNN. Fortunately, we observe that intra-coded blocks are mostly located at key frames or alternative-reference frames, which are frequently selected as anchor points due to their high degrees of reference (see §5.2).

Cache management. To insert/evict the super-resoluted frames to/from cache, we extend the *reference frame buffer* featured in VP9. It contains up to seven decoded frames in memory, and evicts a frame when the reference count over successive frames becomes zero. We extend each element of the buffer to contain a super-resoluted version of the original frame and apply the same eviction policy to both types of frames.

Algorithm 1 Anchor Point Selection

```
1:  function Get-Anchor-Point-Set(Chunk, DNN, VQ_T)
2:      {FQ} = [], {AP'} = [], VQ = 0
3:      {F} ← Get-Frame-From-Video(Chunk)
4:      for f in {F} do
5:          {FQ} += Run-SR-Codec(Chunk, DNN, f)
6:      while VQ(DNN({F})) - VQ > VQ_T do
7:          AP ← Select-New-Anchor({F}, {AP'}, {FQ})
8:          {AP'} += AP
9:          VQ ← Run-SR-Codec(Chunk, DNN, {AP'})
10:     return {AP'}
```

- **Get-Frame-From-Video:** Return all frames of a chunk.
- **Run-SR-Codec:** Apply a DNN to an anchor point set (`f`, `{AP'}`), and return the resulting quality.
- **Select-New-Anchor:** Use Equation 2 to select the anchor point that results in the highest video quality.

5.2 Anchor Point Selection

Goal & Challenge. We present the design of the anchor point selector whose goal is to select a minimal number of anchor points that guarantees the average quality is within a given threshold compared to that of per-frame super-resolution. The optimization goal can be formulated as:

$$\min_{\{AP\}} |\{AP\}| \;\; where \; \{AP\} \subset \{F\}$$
$$s.t. \; VQ(DNN(\{F\})) - VQ(DNN(\{AP\})) \leq VQ_T$$

where $VQ(DNN(\cdot))$ is the video quality enhanced by a super-resolution DNN; $\{F\}$ is the set of all frames; $\{AP\}$ is the set of anchor points; and VQ_T is the target threshold of quality degradation. Obtaining the quality of an anchor set requires running the actual SR-integrated codec and the search space is on the order of $2^{|frame|}$. Thus, exhaustive search to find the minimal anchor point set that satisfies the quality margin is computationally infeasible.

Quality estimation. To avoid actual quality measurements over all possible anchor point sets, we use approximate quality estimation. Generally speaking, the quality of a non-anchor frame can be affected by multiple distinct anchor points. However, when anchor point placements are sparse, the quality improvement of a non-anchor frame is mostly determined by the most impactful anchor point. For example, when 5-20 anchor points are uniformly distributed for each GOP of 120 frames [34], the quality gain attributed to the single most impactful anchor point accounts for 77.8%.

Based on this observation, we approximate the quality gain of a frame that uses a set of anchor points by the maximum quality gain of using a single anchor point in the set:

$$FQ(i|DNN(\{AP\})) \approx \max_{f \in \{AP\}} FQ(i|DNN(f))$$

where $FQ(i|DNN(\cdot))$ is i-th frame quality enhanced by a super-resolution DNN; $DNN(f)$ represents applying a super-resolution

DNN to a single frame. Next, by averaging each approximated frame quality, we can estimate the video quality as:

$$VQ(DNN(\{AP\}) \approx \sum_{i=0}^{|frame|} \max_{f \in \{AP\}} \frac{FQ(i|DNN(f))}{|frame|} \quad (1)$$

where $|frame|$ is the total number of frames within a video. Equation 1 is able to estimate the quality gain of any arbitrary set of anchor points based on the quality measurements of all possible sets of anchor points of size one, reducing the search space of anchor point sets to $|frame|$. We find our quality estimation produces an average error of 0.11 dB in PSNR for our dataset in §7.

Greedy anchor point selection. Next, we use the quality estimation to iteratively select the most effective anchor point, until the resulting video quality difference from per-frame super-resolution falls within the threshold. NEMO uses Algorithm 1 to process each chunk as follows.

The algorithm first measures the quality enhanced by all possible anchor point sets whose size is one (i.e., $FQ(\cdot|DNN(f))$). This is done by running the SR-integrated codec with a single anchor point (line 5). Based on this, it iteratively selects anchor points until the quality requirement is satisfied. For each iteration, it adds a new anchor point that results in the highest video quality using the estimation of Equation 1 (line 7) in the following way:

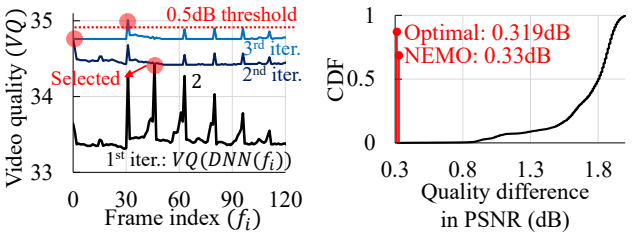
$$\max_{AP \in \{F\}} VQ(DNN(AP \cup \{AP'\})) \approx \max_{AP \in \{F\}} \left(\sum_{i=0}^{|frame|} \max_{f \in \{AP \cup \{AP'\}\}} \frac{FQ(i|DNN(f))}{|frame|} \right) \quad (2)$$

where $\{AP'\}$ is the previously selected anchor points; $\{F\}$ is the set of all frames. Since $FQ(\cdot|DNN(f))$ is already measured in the first step, the equation has linear complexity and can be quickly calculated over each chunk, which consists of around 120 frames.

Next, the algorithm measures the video quality enhanced by NEMO given the current set of anchor points (line 9) compared to that of per-frame super-resolution. Finally, when the quality difference falls within the given threshold (i.e, VQ_T), it records the anchor points into a cache profile, in which 1 bit per frame is used to describe whether a frame is anchor frame or not (line 6); this amounts to about 0.3 KB per minute of video. In this study, we use VQ_T = 0.5, guaranteeing that the quality degradation is under 0.5 dB in PSNR unless otherwise noted.

The algorithm adapts to the video's inherent characteristics discussed in §4.3. First, it is likely to select frames as anchor points with higher degree of reference because such frames deliver larger impact on quality as represented by the inner max selection (max_f). Next, for videos that exhibit lower temporal redundancy, it selects more anchor points to satisfy the quality margin by compensating for the rapid cache erosion.

Illustrative example. Figure 9 illustrates the anchor point selection process from an example drawn from our dataset. For each iteration, the algorithm uses the quality measurements ($VQ(DNN(f_i))$) to calculate Equation 2 for selecting an anchor point that gives the best quality. In the first iteration, it selects frame #46 (alternative reference frame) as an anchor point. Next, frames #0 (key frame) and #31 (alternative reference frame) are added in iterations 2 and 3



(a) Video quality per iteration (b) Exhaustive search

Figure 9: Case study: Anchor point selection

respectively. The process takes 59.6 seconds to complete on a single machine, but can be trivially parallelized.

We compare our result with that of an exhaustive search. Figure 9(b) shows the cumulative distribution function (CDF) of the quality difference from per-frame SR over all possible anchor point sets of size 3. Computing $\binom{128}{3}$ such cases takes 13.2 hours. However, 99.996 % of them fail to meet the quality requirement (0.5 dB), and the PSNR difference between the optimal and NEMO's profile is 0.011 dB, demonstrating the effectiveness of the algorithm.

5.3 Adapting to Devices and Contents

Problem & Goal. To support online video streaming, video super-resolution must be processed at real-time (i.e., 30 fps). However, Algorithm 1 selects a different number of anchor points for each chunk to bound the quality loss within a given threshold. On top of this, mobile devices are heterogeneous with widely varying computing capacities. For example, an entry-level mobile GPU (Qualcomm Adreno 512) has x4.8 less computing capacity that that of a high-end one (Qualcomm Adreno 640).

Providing multiple options. To enable real-time processing under these device- and video-specific constraints, we provide multiple performance options at the server side. For each performance option, we use a separate DNN with varying quality and computing requirement: 'Low', 'Medium', 'High'; the sizes of DNNs range from 118 KB to 1085 KB, and their layer and channel configurations are available at [23]. We then run Algorithm 1 to produce a cache profile for each DNN.

Providing guidelines for mobile devices. Each mobile device should select one of the given options that best suits its computing capacity. For this, NEMO's service provider provides a guideline in selecting performance options for mobile devices by utilizing measurements from a device pool. The device pool is set up to test-run multiple options for each mobile processor type (e.g., Qualcomm Snapdragon 845, 855, and 865); alternatively, mobile testing environments provided by cloud services can be used (e.g., Amazon device farm [5]).

Every processor needs to find the right option for each video. However, testing every option for each video is not a scalable approach as there are a large number of videos. Instead, each device runs an option for a sample video once to obtain the latency of processing an anchor point and a non-anchor frame for the device. Using these results, we can estimate the overall processing latencies of other videos using various options by the following equation:

$$Latency = |AP| \times T(AP) + |None\_AP| \times T(None\_AP)$$

Component	Lines of Code (LoC)	Changed
SR-integraded codec	303K lines of C/C++	2.04% (6182)
Video player	132K lines of Java/C++	0.23% (302)
Anchor point selector	1758 lines of Python	- (1758)
DNN trainer	1163 lines of Python	- (1163)

Table 1: NEMO implementation (Lines of Code)

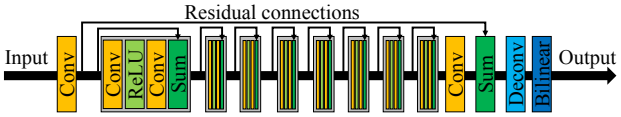


Figure 10: Super-resolution DNN architecture

where $|AP|$, $|None\_AP|$, $T(AP)$, and $T(None\_AP)$ represent the number of anchor points, the number of non-anchor frames, latency of an anchor point, latency of a non-anchor frame, respectively, within a video. In particular, we iterate this estimation over each video chunk. For each mobile processor, an option that delivers the highest quality while meeting the real-time constraint is selected. The manifest file contains this information to guide mobile devices of their selections. Note, this is done offline and incurs onetime cost for each pair of processor type and DNN quality option.

Setting the quality threshold. In Algorithm 1, there is a conflict between setting a quality threshold and guaranteeing the real-time processing. As temporal localities widely vary across chunks, a low quality threshold (e.g., 0.5 dB) may cause a few chunks to have a disproportionately high number of anchor points, which cannot run in real-time. For Youtube videos, as content creators often use numerous editing techniques (e.g. scene transitions, fast forwards, and scene cuts) to make videos more compact, a small number of outlier chunks have extremely low temporal redundancy. Alternatively, a quality threshold can be set conservatively to achieve real-time processing even for the chunk with the smallest amount of temporal redundancy. However, this approach degrades the qualities of all the other chunks.

Instead, to achieve both real-time processing and high-quality, we set a lower quality margin of 0.5 dB but upper-bound the number of anchor points for the outliers: 8 for 'Low', 16 for 'Medium' and 'High', where GOP is set to 120; we set the tighter upper bound for the 'Low' option to support entry-level smartphones. For our Youtube dataset in §7, we observe that there are only a few outliers (9.6 %) and that applying the upper bound to them minimally degrades the video quality (0.03 dB on average).

Energy profiles. We realize that some users could be more battery-conscious than others. To accommodate diverse user preferences, the service provider can offer multiple battery-performance options using multiple different quality thresholds (VQ_T). This also allows users to dynamically adjust the tradeoff between quality and energy consumption depending on the battery level.

6 IMPLEMENTATION

We implement NEMO by extending libvpx [15] and ExoPlayer [14]. Table 1 shows the lines of code (LoC) modified.

SR-integrated codec. We extend libvpx (version 1.7.0) [15], which is a reference software implementation of VP8/9. For DNN inference on mobiles, we use Qualcomm Snapdragon Neural Processing Engine (SNPE) SDK (version 1.40) [25], which provides hardware-accelerated DNN operations for Qualcomm system-on-chips (SoCs). Among three heterogeneous processors (CPU, GPU, NPU) inside the SoC, we execute the DNN on the mobile GPU with half-precision mode (float 16). This is because the CPU is far too slow to support real-time processing and the NPU only provides the quantized integer mode (8-bit) that greatly degrades the quality. Next, to obtain a super-resoluted frame, we convert a decoded YUV420p frame into a RGB888 version, apply a SR DNN to it, and convert the super-resoluted frame back to the original format. We apply the color space conversions because the DNN requires each channel of an input image to have the same shape (e.g., [width, height, #channel]), but in the YUV420p format, the U, V components are four times smaller than the Y component. In addition, we optimize the color space conversions using NEON vectorized instructions [7] available on mobile CPUs.

Video Player. We modify Exoplayer VP9 extension (version 2.9.6) [14] in two ways. First, we make the player to use the SR-integrated codec instead of the default libvpx codec. In addition, the new codec APIs are called to initialize a cache profile and a DNN. Next, we set the maximum buffer size to 60 to ensure smooth rendering. Inside the player, the buffer is shared by the decoding and the rendering thread, which run asynchronously. With the increased buffer size, the latency of an anchor point is sufficiently absorbed by frame buffering (or amortized over successive non-anchor frames.)

Super-resolution DNN. We adopt the DNN architecture and the training methods of NAS [61] but modify the architecture in two ways to reduce the computing/memory usage. As illustrated in Figure 10, the DNN is composed of convolution layers with residual connections followed by a upsampling layer. In NEMO, we substitute sub-pixel convolution with deconvolution because the former causes an out-of-memory error on mobile GPUs. Next, we remove the last convolution layer of the NAS DNN, because it causes large latency on mobile devices but minimally contributes to the output quality.

7 EVALUATION

We evaluate NEMO by answering three questions:

- Does NEMO enable real-time super-resolution on heterogeneous mobile devices in an energy-efficient and temperature-friendly manner?
- Does NEMO improve the QoE of adaptive streaming for mobile users?
- How does each component of NEMO contribute to the overall performance?

Mobile Devices. We use three local devices we purchased and four remote devices from Amazon device farm [5]. For local devices, we use an entry-level smartphone (Xiaomi Redmi Note7), a high-end smartphone (Xiaomi Mi9), and a tablet (LG GPad 5) for our experiments. Table 2 presents their specifications. We use Xiaomi Mi9 as the default device unless otherwise noted. We use the Monsoon High Voltage Power Monitor [21] for measuring the energy consumption, and the FLIR ONE thermographic camera [13] for measuring the surface temperature. During all measurements, we set the screen brightness to its maximum.

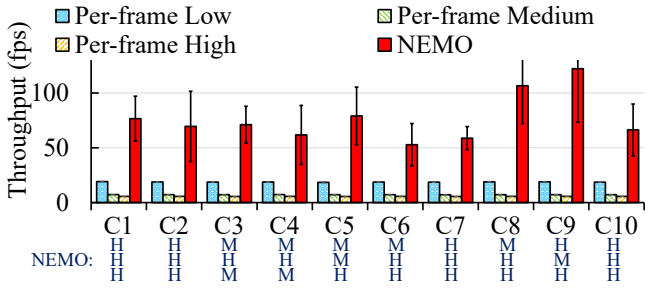


Figure 11: Throughput comparison (240p input)

Model name	Processor	Release	DNN	NEMO
Local mobile devices				
Xiaomi Note7	SDM 660	Jan., 2019	Low	55.1fps (x9.6)
Xiaomi Mi9	SDM 855	Mar., 2019	High	64.5fps (x10.9)
LG GPad5	SDM 821	Mar., 2019	Med.	48.4fps (x11.9)
Amazon device farm				
Samsung A70	SDM 675	Mar., 2019	Low	71.8fps (x13.6)
Samsung Note8	SDM 835	Sep., 2017	Med.	61.2ps (x11.1)
Samsung S6 Tab	SDM 855	July., 2019	High	74.4fps (x10.5)
Samsung S10+	SDM 855	Mar., 2019	High	70.3fps (x10.6)

Table 2: Benchmark on multiple mobiles (SDM: Qualcomm Snapdragon Mobile)

Video dataset. We use 4K videos from the top ten popular categories [20] on YouTube: 'Product review' (**C1**), 'How-to' (**C2**), 'Vlogs' (**C3**), 'Game play' (**C4**), 'Skit' (**C5**), 'Haul' (**C6**), 'Challenges' (**C7**), 'Favorite' (**C8**), 'Education' (**C9**), 'Unboxing' (**C10**). From each category, we select three videos among the top ten most viewed content that supports 4K at 30fps and are at least 5 minutes long [23]. For diversity, we select all three videos from distinct creators. For adaptive streaming, we transcode them into multiple bitrate versions using the VP9 codec as per Wowza's recommendation [30]: {512, 1024, 1600, 2640, 4400}kbps at {240, 360, 480, 720, 1080}p resolutions. The GOP size is 120 (4 sec). We use raw 1080p videos as reference for measuring PSNR. Unless noted otherwise, our evaluation uses the first five minutes of each video.

Baseline. We compare NEMO with the two baseline approaches: 1) *Per-frame DNN* applies a super-resolution DNN in a per-frame basis, and 2) *No DNN* applies bilinear up-sampling instead of super-resolution. We use three different quality DNNs ('Low', 'Medium', 'High') whose full specifications and device matchings are available at [23]. Like in NAS [61], our clients apply DNNs to up-scale {240, 360, 480}p videos to 1080p.

7.1 SR-Integrated Codec and Player

We use four metrics to evaluate NEMO: 1) video processing throughput (i.e., frames per second), 2) resulting video quality, 3) energy consumption, and 4) device temperature.

Throughput improvement. Figure 11 shows the average video processing throughput with NEMO for upscaling 240p to 1080p on Xiaomi Mi9. NEMO selects different quality DNNs depending on the content (§5.3), as labeled below the x-axis (H: High, M: Medium). The error bars show the standard deviation. NEMO significantly improves the processing throughput by x11.5 on average compared to per-frame super-resolution. NEMO achieves 45.1-120.2 fps, whereas all versions of per-frame DNNs violate the real-time constraint.

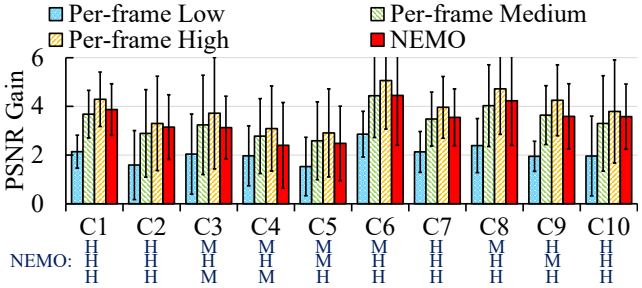


Figure 12: Quality gain comparison (240p input)

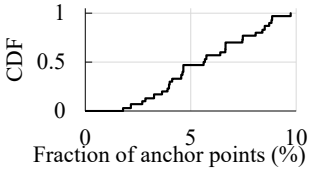


Figure 13: Fraction of anchor points

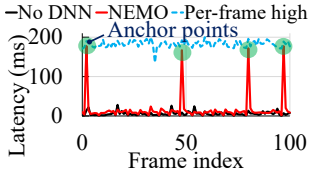


Figure 14: Per-frame latency

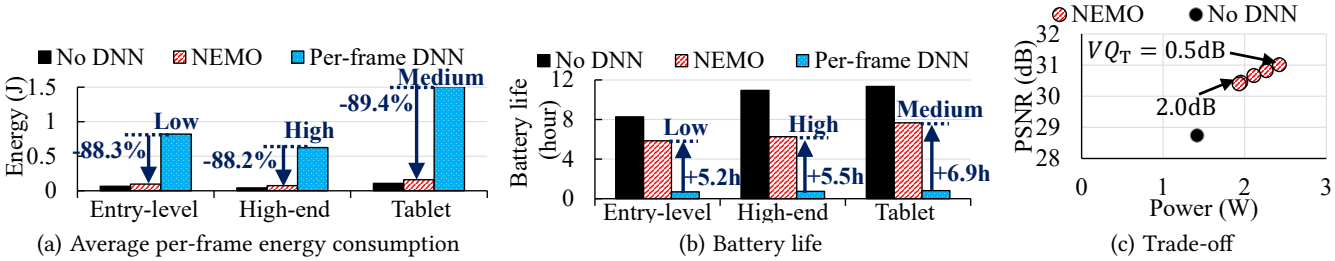
This is because NEMO selects a small fraction of frames as anchor points as shown in Figure 13 which shows the CDF of the fraction of frames that are anchor points within each video in our dataset. The fraction of anchor points ranges between 1.79% and 9.74%, resulting in throughput differences. For example, C6 'Haul' gives 52.8 fps, whereas C8 'Favorite' results in 106.6 fps.

Table 2 shows the throughput on seven different mobile devices. We use the 'Education' video [33] which shows median quality gain amongst the video contents. The table shows NEMO consistently achieves real-time throughput and delivers large throughput improvement (x9.6-x13.6) on heterogeneous devices due to quality adaptation (§5.3). Figure 14 shows the per-frame latency. With NEMO, only the anchor points show large processing latencies, while non-anchor frames are processed very fast, demonstrating significant savings in computation.

Quality. Figure 12 shows the video quality improvement in PSNR compared to the original video, where PSNR is measured using the YUV420p color space. The absolute PSNR of NEMO (not shown in figure) ranges from 30.24 to 44.31 dB. NEMO consistently delivers large quality improvements (0.99-6.35 dB). NEMO's average difference in quality compared to its per-frame SR counterparts is between 0.04 dB and 0.54 dB (0.41 dB on average). NEMO delivers better quality improvements than 'Per-frame Medium' and/or 'Per-frame Low'; e.g., for 'Unboxing' (C10), it outperforms the low- and the medium-quality baseline by 1.62 dB and 0.28 dB, respectively.

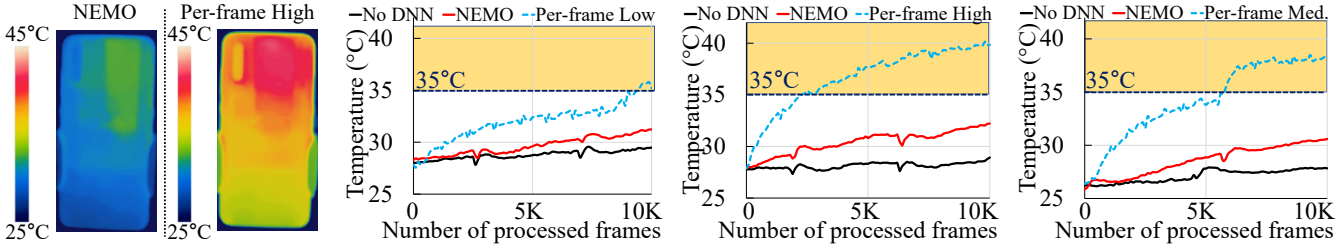
Energy efficiency. Figures 15 (a) and (b) show the average per-frame energy consumption and the expected battery life of an entry-level smartphone, a high-end smartphone, and a tablet for the 'Education' video [33]. NEMO uses 'Low', 'Medium', and 'High' quality DNNs for the entry-level smartphone and high-end smartphone, and tablet, respectively. Compared to per-frame super-resolution, NEMO drastically reduces the energy consumption by 88.3% for the entry-level, 88.2% for the high-end, and 89.4% for the tablet. The expected battery life of NEMO increases by 5.2-6.9 hours. However, compared to traditional video streaming without DNN inference, NEMO still consumes 47.9-74.6% more energy.

Trade-off between power consumption and quality. NEMO allows users to adjust the trade-off by switching cache profiles with



(a) Average per-frame energy consumption (b) Battery life (c) Trade-off

Figure 15: Energy consumption and battery life comparison for 'Education' content (240p input)



(a) Infra-red images (b) Entry-level smartphone (c) High-end smartphone (d) Tablet

Figure 16: Surface temperature comparison for 'Education' content (240p input)

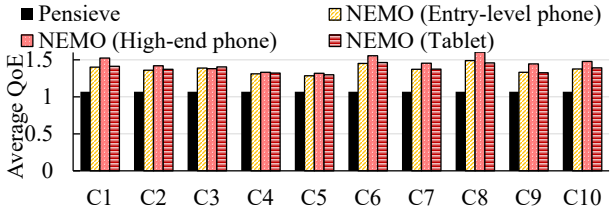


Figure 17: QoE improvement on multiple mobiles

different quality thresholds. Figure 15(c) shows the trade-off with different thresholds for the 'Education' video [33]. Compared to the default threshold setting (i.e., 0.5 dB), the power consumption is reduced by 7.0-25.5 % when the quality is sacrificed by 0.2-0.6 dB. Even with a threshold of 2.0 dB, NEMO delivers a quality improvement of 1.6 dB.

Heat dissipation. Figure 16(a) shows thermographic images of the high-end phone after processing 10,000 frames using NEMO and using the 'Per-frame High', which clearly illustrates the benefit of NEMO. Figures 16 (b), (c) and (d) show the changes in surface temperatures of the hottest point of an entry-level smartphone, a high-end smartphone, and a tablet, respectively, while processing 10,000 frames of a video. The dotted horizontal line is drawn at 35°C, the temperature at which users start to feel discomfort [49, 52]. While per-frame super-resolution resulted in rapid increases in temperature, crossing the 35°C threshold across all devices, NEMO remains below the threshold. For the high-end smartphone, 'Per-frame High' even exceeds 40°C, where users can feel pain due to the excess amount of heat.

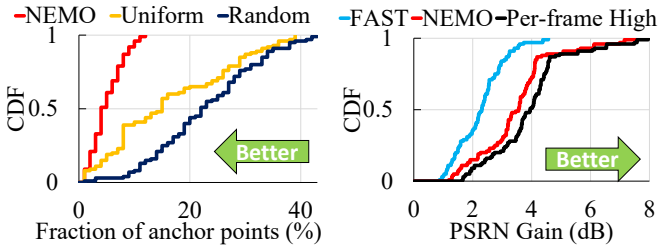
7.2 NEMO vs. Existing Video Delivery

QoE improvement. NEMO enables real-time video super-resolution on mobile devices and thus improves the quality of experience (QoE) of mobile streaming clients. To quantify this, we use real 3G and broadband network traces used in Pensieve [56]. We then filter out the traces whose average bandwidth is higher then 4.4 Mbps to avoid cases where adaptive streaming does not deliver any benefits. The average bandwidth of the network traces is 1.3Mbps. To run adaptive streaming on the given traces, we extend Pensieve's simulator to implement integrated ABR used in NAS [61] that simultaneously streams a DNN and a video. We use the standard QoE metric, used in prior studies [56, 61, 62], that consists of 1) the selected bitrate of each chunk, 2) rebuffering time, and 3) the quality difference between successive chunks. To account for quality improvement due to super-resolution, we create an inverse mapping function from quality to bitrate and obtain the bitrate that corresponds to the enhanced quality as in NAS [61].

Figure 17 shows the average QoE across the ten content categories. As a baseline, we run adaptive streaming using the Pensieve ABR algorithm without applying super-resolution. NEMO consistently outperforms Pensieve for all devices; on average, it delivers 28.7% better QoE for the entry-level, 35.7% for the high-end, and 29.2% for the tablet across all videos. The QoE improvement of NEMO varies between videos, ranging from 20.1% (C5: Skit) to 49.9% (C8: Favorite) because the super resolution gain is video dependent. Pensieve does not show variability across videos because they do not use super-resolution.

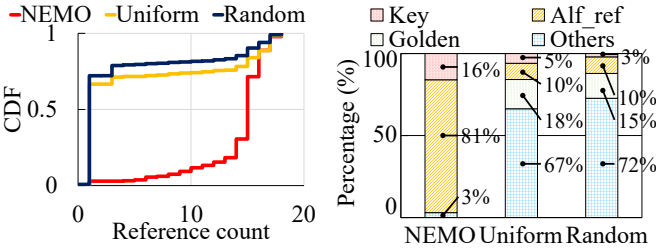
Bandwidth savings. Instead of improving QoE, NEMO can reduce the mobile bandwidth usage while delivering the same QoE. To measure the bandwidth savings, we decrease the bandwidth used by NEMO until NEMO's QoE matches that of Pensieve. On average, NEMO reduces the bandwidth usage by 18.3%, 22.1%, 19.2 % for the entry-level and the high-end smartphone, and the tablet, respectively.

Cost at the server side. We now provide a rough estimate of the server-side cost. Note, the server-side component is not optimized for efficiency, thus the number we provide serves as an upper-bound estimate. First, NEMO trains multiple quality DNNs for each video. The total training time per minute of video is 21.2 minutes on average. Using a Google cloud instance with 12 vCPUs, NVIDIA V100 GPUs, and a 16GB RAM, the cost came out to be $0.75 per minute of video. Next, producing cache profiles costs $0.59 per



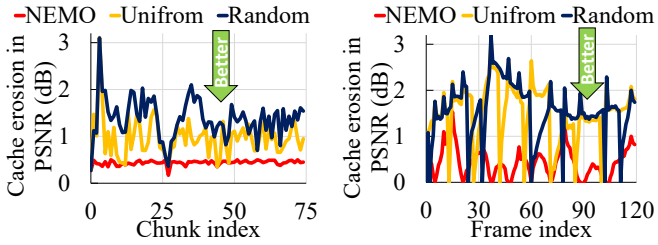
(a) Fraction of anchor points (b) Quality gain

Figure 18: NEMO vs. Baselines



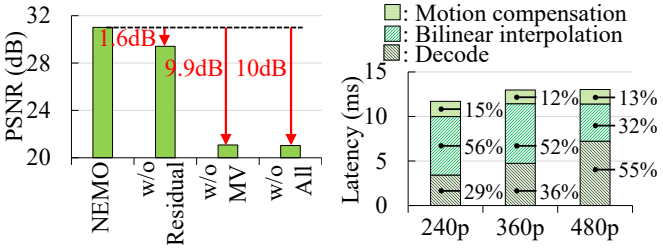
(a) Reference count (b) Frame type

Figure 19: Analysis on anchor points



(a) Per-chunk (b) Per-frame

Figure 20: Analysis on cache erosion



(a) Quality breakdown (b) Latency breakdown

Figure 21: Analysis on SR-integrated codec

minute of video. Finally, NEMO tests multiple performance options on each mobile. Using the Amazon device farm [5] for this costs $0.4 per minute per device.

7.3 Component-wise Analysis

Anchor point selection. NEMO iteratively selects the most effective anchor points and thus minimizes the number of anchor points that meets the quality requirement. To demonstrate this, we compare NEMO with two baselines: 1) *Uniform* that selects uniformly-spaced anchor points 2) *Random* that selects randomly-spaced anchor points. Figure 18(a) shows the CDF of the average fraction of frames that are anchor points for the threshold of 0.5 dB. Here, we do not limit the number of anchor points, as we do for NEMO in §5.3, for a fair comparison. NEMO significantly reduces the number of anchor points by 68.5% and 70.5% on average compared to *Uniform* and *Random*, respectively.

Placing multiple anchor points strategically within each chunk benefits quality. Figure 18(b) shows the CDF of a quality gain. The quality gain of NEMO is within 0.5 dB of per-frame SR. However, compared to an approach that selects only key frames as anchor frames, NEMO delivers 0.3-3.25 dB improved quality. Note, a prior study, FAST [64], proposes this approach. The results demonstrate that the careful selection of anchor points is crucial in guaranteeing the video quality.

Under-the-hood. We now show the characteristics of frames selected as anchor points. Figure 19(a) shows the average reference count of anchor frames and Figure 19(b) shows their makeup broken down into VP9 frame types. We use the 'Education' video [33] as an example. There are three key takeaways: 1) NEMO's anchor points have x2.7 and x3.5 higher reference counts compared to the uniform and random baseline, respectively; 2) while the majority of anchor points (67-72%) in the baselines are normal frames, alternative reference frames and key frames in NEMO make up 97% of anchor frames; and 3) this indicates frame types and dependencies have a strong correlation with the impact of anchor points.

Finally, we demonstrate that NEMO effectively manages cache erosion. Figures 20 (a) and (b) show the per-chunk and the per-frame cache erosion of a GOP, respectively. The baselines use the same number of anchor points as in NEMO. NEMO bounds the average cache erosion of each chunk within 0.5 dB, whereas other baselines show larger cache erosion. The uniform and random baseline are 0.52 dB, 0.94 dB worse than NEMO on average, respectively.

Non-anchor frame processing. The SR-integrated codec leverages motion vectors and residuals to reuse cached high-resolution frames for up-scaling a non-anchor frame. We quantify how each contributes to the resulting quality by excluding them one at a time in Figure 21(a). The result indicates that excluding any information greatly degrades the quality by 1.6 dB (without residual) and 9.9 dB (without motion vector) in PSNR. Finally, Figure 21(b) shows the breakdown of latency in processing non-anchor frames, where we use use 1, 2, 2 threads for 240p, 360p, 480p videos, respectively. Decoding a frame takes between 3.4 to 7.2 ms and NEMO adds between 5.8 to 8.3 ms of latency depending on the resolution, enabling real-time processing.

8 DISCUSSION

Supporting various codecs. NEMO is built upon the software VP9 codec (libvpx). In practice, there are several video codecs beside VP9 (e.g., H.26x [17, 18], VVC [27], AV1 [8]). They differ in low-level compression algorithms (e.g., the size of block, the number of reference frame, the method of motion compensation, and so on) and provide different levels of compression efficiency. Despite the differences, they commonly encode *reference index*, *motion vector*, and *residual* into a compressed video, which are essential information to transfer neural super-resolution outputs to non-anchor frames. Thus, while we only use VP9 to validate NEMO's design, we believe the design of SR-integrated codec is generic enough to accommodate different types of codecs.

Handling resource contention at mobile. When selecting anchor points for each video, NEMO assumes a mobile GPU is solely

used for the NEMO player. However, if multiple applications share the GPU resource, the processing throughput of the DNN can decrease, and thus NEMO may fail to process super-resolution (SR) at real-time. To handle this, several methods can be jointly or individually applied to dynamically adjust the processing throughput at run-time. First, NEMO can skip neural-SR for some anchor points to meet the real-time constraint. Unless they are key frames, cached high-resolution frames can be reused for quality enhancement. Next, as in prior work [41, 61], a SR DNN can be designed to by-pass its partial layers/filters to adapt its computing complexity to available resource budget. Lastly, multiple DNNs with varying qualities can be dynamically switched at the runtime. For future work, we will investigate an optimal approach for handling resource contention.

Supporting 4K video streaming. To support real-time neural-enhanced video streaming, the latencies of anchor points must be amortized by successive non-anchor frames. But, as we target for higher resolutions, the latencies of non-anchor frames also increase and limits real-time processing. For example, for up-scaling 480p video to 2160p (i.e., 4k), a non-anchor frame takes an average of 28.1, 23.9, 17.9 ms for an entry-level smartphone [32], a tablet [19], and a high-end smartphone [31], respectively. As a result, we observe that NEMO cannot support 4K on the entry-level smartphone and the tablet (i.e., cannot amortize the latencies of anchor points), and can only process the 'Low' quality DNN on the high-end smartphone. Currently, NEMO processes non-anchor frames using mobile CPUs, we believe it can be further accelerated by leveraging mobile GPUs/DSPs, which we leave as future work.

9 RELATED WORK

Caching DNN intermediate results. DeepCache [59], DeepMon [45] and CBinfer [38] apply caching and reuse outputs of earlier convolution layers. The key observation is that the early few convolutional layers contribute the largest amount of computation and latency within object classification DNNs. In contrast, super-resolution DNNs enlarge the intermediate outputs as the layers progress in order to recover a high-resolution image. Thus, the majority of computations occur at the last few layers (see §4.1). Based on this observation, NEMO caches the outputted high-resolution frame instead of the intermediate outputs. In reusing the cached outputs, the prior studies match the image blocks only between two consecutive frames. In contrast, NEMO utilizes fine-grained frame dependencies among all frames which have higher (quarter-pixel) precision. This information is embedded in a compressed video.

Accelerating super-resolution (SR) DNNs. Recent mobile devices provide a multitude of heterogeneous processors, such as CPUs, GPUs and DSPs. Inspired by this, MobiSR [54] simultaneously utilizes multiple processors to accelerate image super-resolution on mobile. However, MobiSR does not measure or consider energy consumption and uses CPUs that are less energy-efficient compared to GPUs/DSPs in DNN computations [24]. In contrast, NEMO reduces both the power consumptions and latencies for SR DNNs. NEMO also targets video super-resolution unlike MobiSR which is designed for image super-resolution.

Caching SR outputs. FAST [64] presents a preliminary design that applies neural super-resolution only to the first frame within a GOP and reuses the result. However, this greatly degrades the video quality in an adaptive streaming setting, as we demonstrate in §4.3 and §7.3. FAST is not designed for and does not support video streaming on commodity mobile devices. Unlike NEMO, FAST does not offer a full system implementation, but provides a preliminary evaluation using MATLAB. In contrast, NEMO proposes an SR-integrated codec that enables real-time video SR on commodity mobile devices and integrates it with a commercial video player. In addition, NEMO carefully selects anchor points that deliver the largest quality improvements and provides quality guarantees.

Video streaming optimization. A large body of work has been devoted to improving video streaming. First, for adaptive streaming which NEMO targets, prior studies optimize bitrate/server selections [55, 56, 58, 62] and leverage a centralized video control plane [43, 46, 47]. While they focus on fully utilizing network resource, NEMO uses client computing resource for applying neural video enhancement. Secondly, for live streaming, previous studies reduce the mismatches between a codec and a transport protocol for quickly adapting to bandwidth fluctuations [42, 65] and optimize a codec to enable live 4K encoding on mobiles [35]. Since NEMO targets on-demand contents delivered by adaptive streaming, their efforts are orthogonal to our approaches. We believe that NEMO can be also applied on top of these systems to further improve the live video quality.

Lastly, several studies [40, 44, 50, 61] incorporate neural super-resolution with video streaming applications. They commonly apply computationally expensive DNNs to every frame. As a result, NAS [61], LiveNAS [50], and Dejavu [44] require powerful desktop/server-class GPUs. PARSEC [40] targets mobile streaming, but can only run with micro-models (or extremely small DNNs). As there is a steep trade-off between the DNN size and the resulting quality (see §3), the mirco-models can only provide limited quality benefit. Also, PARSEC can only apply SR to a fraction of a frame to meet the real-time constraint, but NEMO benefits the entire frame. In contrast to NEMO, prior studies do not optimize for energy consumption although per-frame DNN inference greatly decreases the battery life, as illustrated in §3. NEMO enables real-time super-resolution on mobile devices in an energy-efficient manner by taking advantage of the temporal redundancy within a video.

10 CONCLUSION

NEMO is the first video delivery system that enables real-time video super-resolution on commodity mobile devices. Compared to per-frame neural super-resolution, it improves the video processing throughput by x11.5, reduces energy consumption by 88.6%, and maintains device temperatures at acceptable levels. NEMO selectively applies neural super-resolution to a small number of anchor points and reuses their outputs to up-scale the remaining frames within a video to deliver a 0.99-6.34 dB improvement in PSNR. To do so while providing quality guarantees, it adapts to different mobile processors and video contents to maximize the efficiency while ensuring real-time processing. To accommodate diverse user preferences in power management, it allows users to balance the trade-off between energy consumption and video quality. Finally, when used in an adaptive streaming context, NEMO improves the quality of experience (QoE) of mobile users by 31.2 % on average in our trace-driven evaluation with real mobile network traces.

ACKNOWLEDGMENTS

We thank anonymous reviewers and shepherds for providing valuable feedback. We also thank Soowon Kang and Sooyoung Park for helping us to use the FLIR one thermographic camera. Sunghwan Kim contributed to the work in the early days of NEMO. This work was supported by Institute for Information & communications Technology Promotion (IITP) grant funded by the Korea government (Ministry of Science and ICT) [No.2018-0-00693].

REFERENCES

[1] 3G Specification. https://www.etsi.org/technologies/mobile/3g/
[2] 4G Specification. https://www.etsi.org/technologies/mobile/4g/
[3] 5G Specification. https://www.etsi.org/technologies/mobile/5g/
[4] Akamai Mobile CDN. https://www.akamai.com/us/en/resources/mobile-cdn.jsp.
[5] Amazon Device Farm Official Website. https://aws.amazon.com/device-farm/?nc1=h_ls.
[6] Apple HLS Specification. https://developer.apple.com/streaming/.
[7] ARM Neon Overview. https://developer.arm.com/architectures/instruction-sets/simd-isas/neon.
[8] AV1 Specification. https://aomedia.org/av1-features/get-started/.
[9] Big Buck Bunny. https://peach.blender.org/.
[10] Brandwatch Article about "54 Fascinating and Incredible YouTube Statistics". https://www.brandwatch.com/blog/youtube-stats/.
[11] Brightcove Report about "Q3 2019 Brightcove Global Video Index". https://www.brightcove.com/en/video-index/.
[12] Draft VP9 Bitstream and Decoding Process Specification . https://www.webmproject.org/vp9/
[13] FLIR Official Website. https://www.flir.com/.
[14] Google's Exoplayer Official Website. https://developer.android.com/guide/topics/media/exoplayer.
[15] Google's libvpx Official Github Repository. https://github.com/webmproject/libvpx/.
[16] H.264 : Advanced video coding for generic audiovisual services . https://www.itu.int/rec/T-REC-H.264-200305-S/en
[17] H.264 Specification. https://www.itu.int/rec/T-REC-H.264.
[18] H.265 Specification . https://www.itu.int/rec/T-REC-H.265.
[19] LG GPad5 Specifications. https://www.gsmarena.com/lg_g_pad_5_10_1-9952.php.
[20] Medium report about "Top 10 Most Popular Types of Videos on YouTube". https://mag.octoly.com/here-are-the-top-10-most-popular-types-of-videos-on-youtube-4ea1e1a192ac.
[21] Monsoon Official Website. https://www.msoon.com/.
[22] MPEG-DASH Specification. https://dashif.org/.
[23] NEMO's official Github Repository. https://github.com/kaist-ina/nemo.
[24] Qualcomm Article about the Performance of Tensorflow on Mobiles. https://www.qualcomm.com/news/onq/2017/01/09/tensorflow-machine-learning-now-optimized-snapdragon-835-and-hexagon-682-dsp.
[25] Qualcomm Snapdragon Neural Processing Engine Official Website. https://developer.qualcomm.com/docs/snpe/index.html.
[26] Statista Report about "Mobile Share of Global Digital Video Plays from 3rd Quarter 2013 to 2nd Quarter 2018". https://www.statista.com/statistics/444318/mobile-device-video-views-share/.
[27] VVC Specification. https://mpeg.chiariglione.org/standards/mpeg-i/versatile-video-coding.
[28] Webm Official Website. https://www.webmproject.org/.
[29] Wowza CDN for Mobile Video Streaming. https://www.wowza.com/docs/using-wowza-cdn-with-wowza-streaming-engine.
[30] Wowza's DASH bitrate recommendation. https://www.wowza.com/docs/how-to-encode-source-video-for-wowza-streaming-cloud.
[31] Xiaomi Mi9 Specifications. https://www.gsmarena.com/xiaomi_mi_9-9507.php.
[32] Xiaomi Redmi Note7 Specifications. https://www.gsmarena.com/xiaomi_redmi_note_7-9513.php.
[33] YouTube dataset (Education 1). https://www.youtube.com/watch?v=0eaf6bUMd4U.
[34] YouTube dataset (Unboxing). https://www.youtube.com/watch?v=l0DoQYGZt8M.
[35] Ghufran Baig, Jian He, Mubashir Adnan Qureshi, Lili Qiu, Guohai Chen, Peng Chen, and Yinliang Hu. 2019. Jigsaw: Robust live 4k video streaming. In *The 25th Annual International Conference on Mobile Computing and Networking*. 1–16.
[36] Athula Balachandran, Vyas Sekar, Aditya Akella, Srinivasan Seshan, Ion Stoica, and Hui Zhang. 2013. Developing a predictive model of quality of experience for internet video. *ACM SIGCOMM Computer Communication Review* 43, 4 (2013), 339–350.
[37] Jose Caballero, Christian Ledig, Andrew Aitken, Alejandro Acosta, Johannes Totz, Zehan Wang, and Wenzhe Shi. 2017. Real-time video super-resolution with spatio-temporal networks and motion compensation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 4778–4787.
[38] Lukas Cavigelli, Philippe Degen, and Luca Benini. 2017. Cbinfer: Change-based inference for convolutional neural networks on video data. In *Proceedings of the 11th International Conference on Distributed Smart Cameras*. 1–8.
[39] Zhibo Chen, Jianfeng Xu, Yun He, and Junli Zheng. 2006. Fast integer-pel and fractional-pel motion estimation for H. 264/AVC. *Journal of visual communication and image representation* 17, 2 (2006), 264–290.
[40] 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)*.
[41] Biyi Fang, Xiao Zeng, and Mi Zhang. 2018. Nestdnn: Resource-aware multi-tenant on-device deep learning for continuous mobile vision. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*. 115–127.
[42] 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.
[43] Aditya Ganjam, Faisal Siddiqui, Jibin Zhan, Xi Liu, Ion Stoica, Junchen Jiang, Vyas Sekar, and Hui Zhang. 2015. C3: Internet-Scale Control Plane for Video Quality Optimization.. In *Proceedings of the USENIX Conference on Networked Systems Design and Implementation (NSDI)*, Vol. 15. 131–144.
[44] Pan Hu, Rakesh Misra, and Sachin Katti. 2019. Dejavu: Enhancing Videoconferencing with Prior Knowledge. In *Proceedings of the 20th International Workshop on Mobile Computing Systems and Applications*. ACM, 63–68.
[45] Loc N Huynh, Youngki Lee, and Rajesh Krishna Balan. 2017. Deepmon: Mobile gpu-based deep learning framework for continuous vision applications. In *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*. 82–95.
[46] Junchen Jiang, Vyas Sekar, Henry Milner, Davis Shepherd, Ion Stoica, and Hui Zhang. 2016. {CFA}: A practical prediction system for video qoe optimization. In *13th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 16)*. 137–150.
[47] Junchen Jiang, Shijie Sun, Vyas Sekar, and Hui Zhang. 2017. Pytheas: Enabling Data-Driven Quality of Experience Optimization Using Group-Based Exploration-Exploitation.. In *Proceedings of the USENIX Conference on Networked Systems Design and Implementation (NSDI)*, Vol. 1. 3.
[48] 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.
[49] Soowon Kang, Hyeonwoo Choi, Sooyoung Park, Chunjong Park, Jemin Lee, Uichin Lee, and Sung-Ju Lee. 2019. Fire in Your Hands: Understanding Thermal Behavior of Smartphones. In *The 25th Annual International Conference on Mobile Computing and Networking*. 1–16.
[50] 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 ACM Special Interest Group on Data Communication (SIGCOMM)*.
[51] 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.
[52] JC Lawrence and JP Bull. 1976. Thermal conditions which cause skin burns. *Engineering in Medicine* 5, 3 (1976), 61–63.
[53] Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, et al. 2017. Photo-realistic single image super-resolution using a generative adversarial network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 4681–4690.
[54] 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.
[55] 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 Special Interest Group on Data Communication (SIGCOMM)*.
[56] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. 2017. Neural Adaptive Video Streaming with Pensieve. In *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM)* (Los Angeles, CA, USA). 197–210.
[57] Matthew K. Mukerjee, David Naylor, Junchen Jiang, Dongsu Han, Srinivasan Seshan, and Hui Zhang. 2015. Practical, Real-time Centralized Control for CDN-based Live Video Delivery. In *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM)* (London, United Kingdom). 311–324.
[58] Kevin Spiteri, Rahul Urgaonkar, and Ramesh K Sitaraman. 2016. BOLA: Near-optimal bitrate adaptation for online videos. In *Proceedings of the IEEE International Conference on Computer Communications (INFOCOM)*. IEEE, 1–9.

[59] Mengwei Xu, Mengze Zhu, Yunxin Liu, Felix Xiaozhu Lin, and Xuanzhe Liu. 2018. DeepCache: Principled cache for mobile deep vision. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*. 129–144.

[60] Hyunho Yeo, Sunghyun Do, and Dongsu Han. 2017. How will Deep Learning Change Internet Video Delivery?. In *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*. ACM, 57–64.

[61] 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.

[62] 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 ACM Special Interest Group on Data Communication (SIGCOMM)*.

[63] 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.

[64] 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.

[65] 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.

[66] Ce Zhu, Xiao Lin, and Lap-Pui Chau. 2002. Hexagon-based search pattern for fast block motion estimation. *IEEE transactions on circuits and systems for video technology* 12, 5 (2002), 349–355.

[67] Shan Zhu and Kai-Kuang Ma. 2000. A new diamond search algorithm for fast block-matching motion estimation. *IEEE transactions on Image Processing* 9, 2 (2000), 287–290.