박 사 학 위 논 문
Ph.D. Dissertation

신경망 화질 강화 기반 비디오 스트리밍 연구

Enabling Neural-enhanced Video Streaming

2023

여 현 호 (呂 賢 虎 Yeo, Hyunho)

한 국 과 학 기 술 원

Korea Advanced Institute of Science and Technology

박 사 학 위 논 문

신경망 화질 강화 기반 비디오 스트리밍 연구

2023

여 현 호

한 국 과 학 기 술 원

전기및전자공학부

신경망 화질 강화 기반 비디오 스트리밍 연구

여 현 호

위 논문은 한국과학기술원 박사학위논문으로
학위논문 심사위원회의 심사를 통과하였음

2023년 4월 13일

심사위원장	한 동 수	(인)
심 사 위 원	신 진 우	(인)
심 사 위 원	박 경 수	(인)
심 사 위 원	이 성 주	(인)
심 사 위 원	황 의 종	(인)
심 사 위 원	Junchen Jiang	(인)

Enabling Neural-enhanced Video Streaming

Hyunho Yeo

Advisor: Han, Dongsu

A dissertation submitted to the faculty of
Korea Advanced Institute of Science and Technology in
partial fulfillment of the requirements for the degree of
Doctor of Philosophy in Electrical Engineering

Daejeon, Korea
April 13, 2023

Approved by

Han, Dongsu
Professor of Electrical Engineering

The study was conducted in accordance with Code of Research Ethics[1].

[1] Declaration of Ethical Conduct in Research: I, as a graduate student of Korea Advanced Institute of Science and Technology, hereby declare that I have not committed any act that may damage the credibility of my research. This includes, but is not limited to, falsification, thesis written by someone else, distortion of research findings, and plagiarism. I confirm that my thesis contains honest conclusions based on my own careful research under the guidance of my advisor.

DEE 여현호. 신경망 화질 강화 기반 비디오 스트리밍 연구. 전기및전자공학부 . 2023년. 84+v 쪽. 지도교수: 한동수. (영문 논문)
Hyunho Yeo. Enabling Neural-enhanced Video Streaming. School of Electrical Engineering . 2023. 84+v pages. Advisor: Han, Dongsu. (Text in English)

초 록

지난 수십 년간 인터넷 비디오 스트리밍에 대한 수요는 폭발적으로 증가하였다. 현재의 비디오 스트리밍 인프라는 두 가지 핵심 기술을 통해 확장성 문제를 성공적으로 처리해 왔다: 1) 서버 측에서는 분산 컴퓨팅 기술 및 콘텐츠 전송 네트워크를 통해 인터넷 규모의 콘텐츠 전송이 가능해졌고, 2) 클라이언트 측에서는 적응형 비트 레이트 스트리밍을 통해 사용자 간 및 시간에 따른 대역폭 변화 문제를 해결했다. 그러나 현재 비디오 스트리밍 인프라는 비디오 화질이 대역폭에 지나치게 의존적이라는 근본적인 한계점을 가지고 있다. 그래서 네트워크가 혼잡해져 대역폭이 부족해지면 사용자의 체감 품질 만족도가 급격히 하락한다.

본 학위 논문에서는 최근 급속히 증가한 연산 능력과 딥러닝의 기술 발전에서 영감을 받아 비디오 스트리밍 품질을 향상할 수 있는 새로운 방법론을 제시한다. 구체적으로 본 학위 논문에서는 클라이언트와 서버의 연산 자원을 활용해 심층 신경망 기반 화질 향상을 적용하여 비디오 스트리밍에서 사용자 체감 품질을 크게 개선할 수 있음을 보인다. 네트워크 대역폭에 크게 의존하는 종래의 스트리밍과는 달리, 제안하는 방법론에서는 비디오 콘텐츠에 심층 신경망 기반 화질 향상을 적용하여 저화질 전송에서 고화질 비디오를 복원한다. 이 방법론은 현재 비디오 스트리밍 인프라를 기반으로 사용자 체감 품질을 극대화할 수 있는 강력한 메커니즘을 제공한다

그러나 심층 신경망 기반 화질 향상을 비디오 스트리밍에 적용하는데 두 가지 근본적인 문제점이 있다. 첫 번째로 심층 신경망 예측의 성능은 새로운 콘텐츠에 대해 신뢰할 수 없다. 특히 상용 애플리케이션 상의 방대한 양의 비디오에 대해 일관된 품질 향상을 보장하기는 굉장히 힘들다. 두 번째로 화질 향상에 사용되는 심층신경망의 컴퓨팅 비용은 상업용 클라이언트 및 서버에 적용하기에 지나치게 비싸다. 이 문제점은 제안된 방법론의 적용 가능한 범위를 심각하게 제한한다. 사용자 기기에서 심층 신경망을 적용하는 경우 (예시: 서버에서 사용자로의 비디오 전송), 상용 모바일 기기에서 실시간으로 실행될 수 없기 때문에 모바일 사용자를 지원하지 못한다. 미디어 서버에서 심층 신경망을 적용할 경우 (예시: 스트리머에서 미디어 서버로의 비디오 전송), 지나치게 큰 컴퓨팅 비용이 발생해 상용 스트리밍 어플리케이션 지원이 불가능하다.

본 학위 논문에서는 이러한 근본적인 문제점을 해결하는 일련의 방법론을 제시한다. 첫째, 심층 신경망을 통한 안정적인 품질 향상을 보장하기 위해 각 콘텐츠에 대해 심층 신경망을 개별적으로 학습시키는 콘텐츠 인식 접근 방식을 제안한다. 이 방법론은 예측할 수 없는 테스트 정확도에 의존하는 대신 심층 신경망의 과적합 속성을 활용하여 예측할 수 있고 정확한 학습 정확도를 제공한다. 다음으로, 심층 신경망의 컴퓨팅 비용을 줄이기 위해 일부 선택된 비디오 프레임에만 심층 신경망을 적용하고, 이 결과를 재사용하여 전체 비디오의 화질을 개선하는 선택적 추론 방식을 제안한다. 이 방법론은 비디오 내에서 방대한 양의 시간적 중복성을 활용하여 심층 신경망의 컴퓨팅 복잡도를 획기적으로 감소시킨다. 마지막으로 본 학위논문에서는 제안된 방법론을 종단 간 시스템으로 구현하여 심층 신경망 기반 화질 향상을 통해 사용자 체감 품질을 획기적으로 향상할 수 있음을 보인다.

핵 심 낱 말 비디오 스트리밍, 딥러닝, 네트워크 시스템

Abstract

Internet video has experienced tremendous growth over the last few decades. Current video delivery infrastructure has been successful in handling scalability challenges with two key technologies: 1) at the server side, distributed computing technologies and content delivery networks (CDNs) have enabled content delivery at the Internet scale; and 2) at the client side, adaptive bitrate (ABR) streaming has addressed the problem of bandwidth heterogeneity and its variations across time and space. However, there is a fundamental limitation that the quality of existing video delivery critically depends on the bandwidth resource. Consequently, user quality of experience (QoE) inevitably suffers when network conditions become unfavorable.

This dissertation proposes an alternative and complementary approach to enhancing video quality, inspired by the ever-increasing computational power and recent advances in deep learning. Our thesis is that *one can substantially improve user QoE in video streaming by neural enhancement based on the client and server computation*. Unlike traditional streaming, which heavily depends on network resources, we apply a deep neural network (DNN)-based quality enhancement on video content to obtain high-definition (e.g., 1080p) video from lower quality transmissions. This provides a powerful mechanism for maximizing QoE on top of the existing video delivery infrastructure.

However, realizing neural-enhanced video streaming poses two fundamental challenges. First, the performance of DNN predictions is unreliable for unseen/new content. Ensuring consistent quality enhancement is especially difficult, which presents a significant barrier to practical deployment. Second, a DNN used for quality enhancement requires too much computation to run on commodity servers and clients. This severely limits its applicability to Internet-scale streaming services. For downstream video streaming, where a DNN runs at clients, it is infeasible to support real-time neural enhancement on mobile devices. For upstream video ingest, where a DNN runs at media servers, it is too costly to support commercial-scale live streaming using a public cloud.

To prove the thesis, this dissertation presents a collection of algorithmic and architectural solutions that address these challenges based on domain-specific insights into DNNs and video streaming. First, we develop a content-aware approach to ensure reliable quality enhancement powered by a DNN. In this approach, we train a separate DNN for each content to exploit the DNN's overfitting property, which delivers reliable and predictable training accuracy instead of relying on unpredictable test accuracy. Next, we reduce the computing cost of a DNN by devising a selective inference approach. In this approach, the DNN is applied only to a few select frames, and the results are then reused to benefit the entire video, by leveraging the vast amount of temporal redundancies within a video. Lastly, we validate our solutions using full-fledged systems to demonstrate significant improvements in user QoE for video streaming.

Keywords Video Streaming, Deep Learning, Networked Systems

Contents

List of Tables

List of Figures

Chapter 1. Introduction

Internet video has experienced tremendous growth over the last few decades and continues to grow rapidly. Video now accounts for 73% of Internet traffic and is expected to quadruple in the next five years [113, 188]. This trend will be further accelerated by the emergence of new media such as augmented reality (AR) and virtual reality (VR), which are projected to increase twenty-fold in five years [113]. At the same time, due to users' steep expectations for quality, delivering a high quality of experience (QoE) has become of paramount importance [100]. To meet the seemingly insatiable demand, current video delivery infrastructure has made onerous efforts to handle the scalability challenges with two key technologies. First, at the server side, distributed computing technologies and content delivery networks (CDNs) have enabled content delivery at Internet scale [178]. Second, at the client side, adaptive bitrate (ABR) streaming has addressed the problem of bandwidth heterogeneity and its variations across time and space. Techniques at both ends have evolved over time to optimize user quality of experience (QoE), as it ultimately impacts the revenue of various stakeholders [101, 214, 117].

Despite these efforts, current video delivery infrastructure still suffers from the fundamental limitation that video quality heavily depends on the bandwidth between servers and clients. When the bandwidth resource becomes scarce, video quality suffers directly [148, 157]. Since video quality is the most important factor affecting user QoE in video streaming [117, 79], quality degradation in turn greatly harms the revenue of video streaming providers [70, 67]. For example, more than 50% of live viewers abandon a stream when video quality suffers more than 90 seconds [36]. Thus, alleviating the tight dependency between network bandwidth and video quality is a critical and time-sensitive problem.

1.1 Limitations of Prior Approaches

Traditional methods for improving video stream quality include using better codecs [73, 74], optimizing adaptive bitrate algorithms [96, 141, 147], selecting better servers and CDNs [93, 166, 205], and coordinating clients and servers through a centralized control plane [167, 175]. These methods focus on how to best utilize network resources, but they suffer from two common limitations.

① Traditional streaming underutilizes powerful computational resources: Modern devices, including commodity servers, desktops, and smartphones, now come equipped with significant computational power. According to market reports [180], around 50% of users watch videos on personal computers, which have considerable computational power. The rest of the users watch videos on mobile devices, which come equipped with power-efficient mobile graphics processing units (GPUs); these GPUs outperform older-generation game consoles such as the Xbox 360 [120]. Additionally, the increasing popularity of artificial intelligence and mobile games, as well as emerging media like virtual and augmented reality, will significantly accelerate this trend. Consequently, there is a large opportunity to trade off computation for reduced bandwidth under network congestion. Unfortunately, the current video delivery infrastructure does not allow for the utilization of powerful computational power. Therefore, when network congestion occurs, the quality of the stream suffers directly.

② Traditional video codecs cannot take advantage of long-term redundancy: Video content contains a significant amount of redundancy, particularly over long timescale; its high-level features contain valuable information that can be leveraged for video coding. For example, meaningful objects recognized

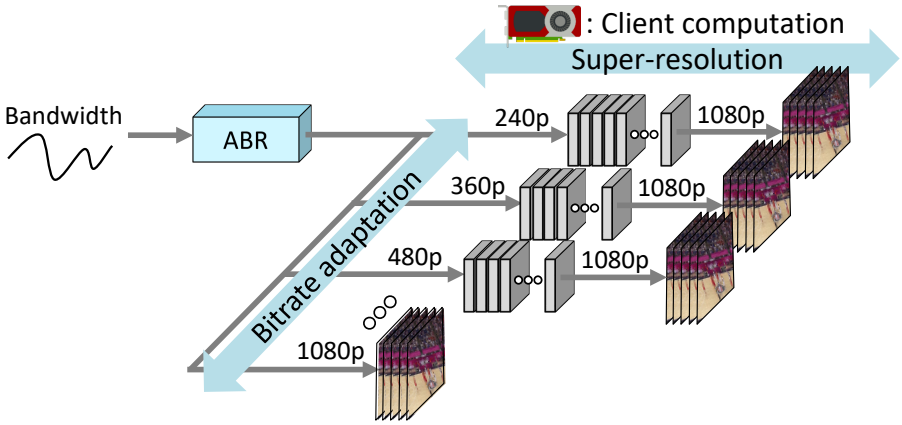


Figure 1.1: A DNN can utilize computing power to maximize user QoE.

by human, such as sport player, stadium and score board, reappear frequently. However, standard video coding techniques, such as MPEG and H.26x, only capture two types of short-term redundancy and lack any mechanisms to leverage the semantics of motion pictures. Spatial redundancy exploits pixel-level similarity within a picture [204]. The intra-frame coding compresses a picture using discrete cosine transform (DCT), quantization, and entropy encoding [131]. Temporal redundancy represents similarities between successive frames. Inter-frame coding encodes the difference between adjacent frames to compresses a motion picture [121].

1.2 New Paradigm: Neural-enhanced Video Streaming

Inspired by the ever-increasing computational power and recent advances in deep learning, this dissertation presents a new paradigm called *neural-enhanced video streaming*, which is an alternative and complementary approach to enhancing user QoE. Unlike traditional streaming, which heavily depends on network resources, we apply a deep neural network (DNN)-based quality enhancement on video *content* utilizing the *client and server computation* to maximize user QoE. In essence, neural-enhanced video streaming takes advantage of two key benefits of a DNN.

① A DNN enables us to leverage computing power to maximize user QoE: A DNN is a computational model that consists of multiple layers of hierarchy; each layer processes the input in a non-linear fashion and delivers its output to the layer above it. Since a DNN is designed to learn high-level abstract features from a complex low level representation of data [103], it can also learn an accurate mapping from low-quality video to a high-quality version. For example, a super-resolution DNN recovers a high-resolution video (e.g., 1080p) from a lower-resolution counterpart (e.g., 240p-720p). As illustrated in Figure 1.1, these characteristics enable us to utilize computing power to maintain high user QoE even when video is transmitted in lower quality due to network congestion. This provides a powerful mechanism for QoE maximization on top of bitrate adaptation.

② A DNN enables us to utilize long-term redundancy for video encoding: A DNN learns high-level features from the entire video clip to effectively recover high-quality video, as depicted in Figure 1.2. In the basketball game shown in the figure, balls, players, and scenery, such as the court and background, appear repeatedly quarter after quarter with small variations. Similarly, such redundancy is also evident within episodes of each TV show, games in sports leagues, and videos from the same YouTube or Twitch streamers. Thus, we argue that a DNN is actually capturing and encoding common

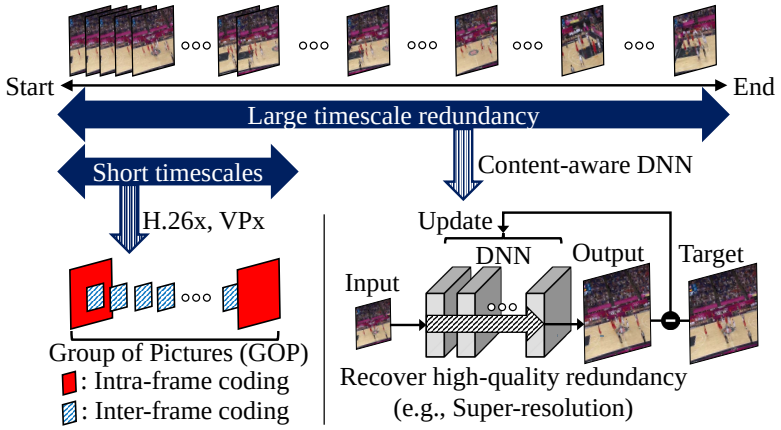


Figure 1.2: A DNN can leverage long-term redundancy for video encoding.

representations spread out over large timescales (i.e., long-term redundancy), providing new opportunities for more efficient video compression.

1.3 Enabling Neural-enhanced Video Streaming

The main contribution of this dissertation is *a collection of algorithmic and architectural solutions that enable neural-enhanced video streaming*. Specifically, we first identify critical challenges in applying neural enhancement to video streaming, provide concrete algorithms and system designs based on the domain-specific insights into deep neural networks and video streaming, and use full-fledged systems to validate that our solutions can significantly improve user quality of experience (QoE) for video streaming.

1.3.1 Key Challenges

To enable neural-enhanced video streaming, one must address two primary challenges.

① Providing reliable quality improvement using a DNN is difficult: It is impractical to develop a universal DNN model that works well across all Internet videos, as the number of video episodes is almost infinite. A single DNN of finite capacity, in principle, may not be expressive enough to capture all of them. Furthermore, there exists a trade-off between generalization and specialization for any machine learning approach: as the model coverage becomes larger, its performance degrades, which is referred to as the "no free lunch" theorem [206]. Even worse, one can generate "adversarial" videos of arbitrarily low quality, given any existing DNN model [140, 182]. This makes the service vulnerable to reduction of quality attacks, presenting a significant barrier for deployment.

② A DNN requires too much computational power to run on commodity servers and clients: A DNN used for quality enhancement is typically 100-1000× more expensive compared to a DNN used for discriminative tasks [161]. This excessive computation severely limits its applicability to Internet-scale streaming services. For video streaming, where clients run a DNN, real-time neural enhancement is infeasible on mobile devices due to impractical delays, unsustainable rates of battery drain, and large amounts of heat dissipation [151] that cause discomfort to users. For video ingest, where media servers run a DNN, applying a DNN to commercial-scale live streaming requires tens of thousands of GPUs, which cost over $169,000 per hour on a public cloud.

1.3.2 Unifying Insights

We address these challenges using domain-specific insights into DNNs and video streaming.

① Content-awareness can ensure consistent quality enhancement across videos: DNNs typically achieve near-zero training error (i.e., overfitting), but the testing error is often much higher due to overfitting [194]. Although the deep learning community has made extensive efforts to reduce this gap [194, 144], relying solely on a DNN's testing accuracy may result in unpredictable performance [140, 182]. To leverage the inherent overfitting property of DNNs, we propose a content-aware DNN that uses a different DNN for each video episode. This guarantees reliable and superior performance.

② Temporal redundancy offers a room for reusing DNN outputs within video: Video contains a significant amount of temporal redundancy, which presents large opportunities to reduce the computational overhead of a DNN by reusing its outputs from selected frames. However, previous studies fall short of efficiently utilizing temporal redundancy to save DNN computations in the context of neural enhancement. To address this, we present selective DNN inference using fine-grained dependencies using information from the video codec.

1.3.3 Proposed Solutions

Based on these key insights, this dissertation develops three full-fledged solutions to address the two aforementioned challenges to enable neural-enhanced video streaming in the context of adaptive streaming and live ingest. Next, we summarize these three key components that constitute this dissertation.

① Neural-enhanced Video Streaming (Chapter §3). Internet video streaming has experienced tremendous growth over the last few decades. However, the quality of existing video delivery critically depends on the bandwidth resource. Consequently, user quality of experience (QoE) suffers inevitably when network conditions become unfavorable. We present NAS [212], a new video delivery framework that utilizes client computation and recent advances in deep neural networks (DNNs) to reduce the dependency for delivering high-quality video. The use of DNNs enables us to enhance the video quality independent to the available bandwidth. We design a practical system that addresses several challenges, such as client heterogeneity, interaction with bitrate adaptation, and DNN transfer, in enabling the idea. Our evaluation using 3G and broadband network traces shows the proposed system outperforms the current state of the art, enhancing the average QoE by 43.08% using the same bandwidth budget or saving 17.13% of bandwidth while providing the same user QoE.

② Neural-enhanced Mobile Streaming (Chapter §4). 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 [210], 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 provides 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 x15.2, reduces energy consumption by 88.7%, and maintains device temperatures at acceptable levels compared to per-frame super-resolution, while ensuring high video quality. Overall, this leads to a 27.5% improvement in quality of experience for mobile users.

③ Neural-enhanced Ingest at Scale (Chapter §5). 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 [213], 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.

1.4 Organization

The rest of this dissertation is organized as follows. Chapter §2 provides primer in video streaming, super-resolution, and DNNs and then presents literature survey related to our work. Chapter §3, §4, and §5 illustrate three key components of this dissertation: 1) NAS addresses the challenge of reliable quality improvement, 2) NEMO addresses the challenge of computing overhead in the context of mobile computing and video streaming, and 3) NeuroScaler addresses the challenge of computing overhead in the context of cloud computing and live video ingest. Chapter §6 provides a summary of the future research directions aimed at enhancing the practicality and wider applicability of neural-enhanced video streaming. Chapter §7 concludes the dissertation by summarizing the key contributions.

Chapter 2. Background & Related Work

In this chapter, we begin with a technical background on video streaming, super-resolution, and deep neural networks (DNNs), which are essential components of neural-enhanced video streaming (§2.1). Next, we provide an overview of recent studies that are relevant to our dissertation. These include the extensive efforts of the networking and system communities to enhance user QoE in video streaming (§2.2) and various techniques for reducing the computational complexity of neural enhancement (§2.3).

2.1 Primer in Video Streaming and Neural Enhancement

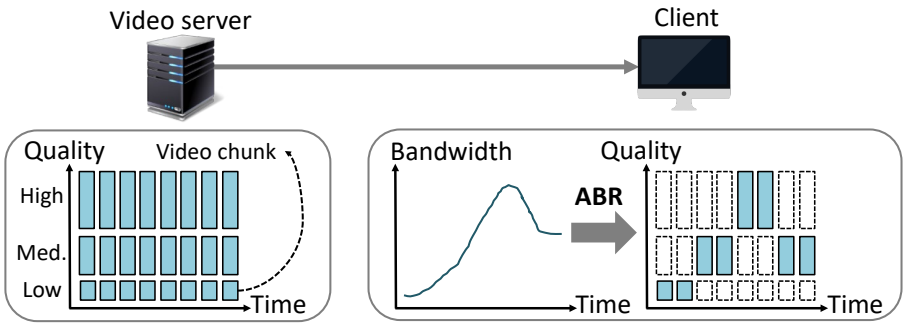


Figure 2.1: Adaptive streaming end-to-end workflow

Adaptive streaming (e.g., Apple's HLS [5], DASH [17]) is designed to handle unpredictable bandwidth variations in the real world. Figure 2.1 shows the overall workflow of traditional adaptive streaming. At the server side, video is encoded into various bitrates (or resolutions) and divided into fixed length chunks, typically 2 – 10 seconds. At the client side, an adaptive birate algorithm (ABR) decides the bitrate for each video chunk. Traditional ABR algorithms select bitrates using heuristics based on the estimated network bandwidth [147] and/or the current size of the client-side playback buffer [193]. Recent ABR algorithms directly optimize a user QoE metric using model predictive control [214] or reinforcement learning [169].

Live streaming consists of the ingest and distribution side. First, at the ingest side, the streamer captures live video and uploads it to a media server. In adaptive streaming [17, 5], the streamer uploads a single video stream, and the media server transcodes it into multiple quality versions. In multi-party video conferencing [78], a broadcaster uploads multiple quality streams using simulcast [77] or scalable video codec (SVC) [63], 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.

Video codecs (e.g., H.26x [73, 74], VPx [76]) 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 [27, 34]. Next, the difference in pixel values between the the reference and the target block, referred to as *residual*, is encoded. Intra-coded blocks follow the same process, except they select blocks within the same frame as reference.

Super-resolution recovers a high-resolution image from a single or multiple *lower resolution* image(s). Super-resolution has been used in a variety of computer vision applications, including surveillance [224] and medical imaging [192], where the original high-quality image/video is not available. Recent studies use DNNs [154, 217, 150, 159, 106] to learn low-resolution to high-resolution mapping and demonstrate a significant performance gain over non-DNN approaches [108, 189].

Scalable DNN is an emerging type of DNN designed to dynamically adapt to computational resource constraints, enabling *anytime prediction* [135]. A shallow and a deep network are used in resource-constrained and -sufficient environments respectively [105, 135]. ISResNeXt [160] alternatively uses a thin and a wide network that adapts to the width (or channels) of a DNN. Scalable DNN has been applied primarily to image classification/detection tasks.

2.2 Improving User QoE in Video Streaming

Improving an ABR algorithm. Several studies focus on designing a better ABR algorithm to maximize user QoE. Traditional ABR algorithms select bitrates using heuristics based on the estimated network bandwidth [147] and/or the current size of the client-side playback buffer [193]. MPC [214] and Pensieve [169] demonstrate that directly optimizing for the desired QoE objective delivers better outcomes than heuristics-based approaches. In particular, Pensieve uses deep reinforcement learning and learns through "observations" how past decisions and the current state impact the video quality. Oboe [97] dynamically adjusts the ABR parameters depending on the network conditions consulting the offline pre-computation result. Fugu [209] extends MPC by introducing a data-driven throughput predictor that learns from the real deployment environment. Pano [126] introduces a quality model that accounts for the unique factors of 360-degree videos, while Sensei [216] proposes a QoE model that incorporates content-dependent quality sensitivity. To enhance the performance of QoE-driven ABR algorithms, both Pano and Sensei utilize application-aware QoE models. Although these algorithms successfully cope with bandwidth variations, they consider neither the effect of client-side quality enhancement nor the dynamics of simultaneously streaming a DNN and video chunks.

Leveraging a video control plane. Prior work [167, 125, 146, 149, 162, 165] identifies spatial and temporal diversity of CDNs in performance and advocate for an Internet-scale control plane which coordinates client behaviors to collectively optimize user QoE. Although they control client behaviors, they do not utilize client computation to directly enhance the video quality.

Optimizing live streaming. Live streaming has two unique features that distinguish it from other streaming types: live viewers experience varying levels of delay, and delay becomes a crucial factor that affects streaming experience. Vantage [184] proposes a video ingest method targeted at applications where users watch videos at different delays. The system selectively re-transmits previously uploaded frames with higher quality, considering the delay statistics, to maximize overall user QoE. The system identifies a sweet spot that balances the QoE of real-time viewers and delayed viewers via selective

re-transmission. On the other hand, Salsify [122] enables a tight integration between the video codec and transport protocol to quickly adapt to available bandwidth, which in turn leads to QoE improvements for live streaming. Our work is complementary to these efforts. In particular, our ingest framework, NeuroScaler, can integrate DNN-based quality enhancement into these systems.

Supporting video processing at scale. Back-end video processing systems have been of growing importance due to the scale required for video encoding. Studies have reported that latency for fast interactive sharing, system efficiency in encoding, scalability and fault tolerance are major issues [139, 123, 198]. SVE [139] presents a backend system for video processing used in Facebook. Atlas [171] presents a high-performance video streaming server based on user-space networking, as used in Netflix. ExCamera [123] uses massive parallelism to enable interactive and collaborative editing. The prior studies focus on solving distributed system problems within a datacenter without changing the clients, whereas our dissertation focuses on the division of work between the servers and clients.

Leveraging neural enhancement. Our research has inspired other studies to utilize DNN-based quality enhancement to enhance the streaming experience. At the ingest side, CloudSeg [202] and LiveNAS [152] have employed neural enhancement to ingest streams for analytic tasks and traditional streaming, respectively. CloudSeg has proposed a training scheme that tailors super-resolution DNNs for analytic tasks, while LiveNAS has presented an online learning framework that enables content-aware DNNs for live streaming. Our contributions to the ingest side, NeuroScaler, can be integrated into these ingest frameworks to improve processing throughput and resource efficiency. At the distribution side, PARSEC [115] and Yuzu [215] have applied neural enhancement to 360-degree panoramic streaming and volumetric streaming, respectively. PARSEC has adopted small micro models for faster inference and efficient transmission and redesigned an ABR algorithm to determine whether to download high-resolution video chunks or generate them from lower-resolution chunks. Yuzu has derived a QoE metric for neural-enhanced volumetric video and utilized it to accelerate super-resolution DNNs in a way that minimizes QoE degradation. Although our work focuses on traditional 2D videos, the concepts of content-aware DNNs and selective inference can be extended to these frameworks to enhance reliability and cost-effectiveness.

Leveraging neural compression. Recent advancements in DNN-based image and video compression techniques have shown great potential, as evidenced by various studies [164, 172, 173, 138, 112, 102, 199, 200, 116]. However, these learnable codecs have a major drawback when compared to conventional codecs: their high computing costs make it impractical to deploy them in real-world scenarios involving a variety of heterogeneous clients and servers. Therefore, our work aims to develop systems that can use traditional codecs while still being capable of incorporating neural codecs in a generic manner.

2.3 Reducing the Overhead of Neural Enhancement

Improving DNN architecture. Numerous studies have been conducted to reduce the computing overhead of DNNs used for quality enhancement. To address this issue, previous research has used three distinct approaches. Firstly, some studies have developed efficient feature extraction [95, 142, 219] or up-scaling layers [119, 191]. Second, ClassSR [156] and MobiSR [161] refer to the difficulty of SR to adjust the capacity of SR network within an image. Lastly, SLS [179] has developed a model compression

technique specifically designed for neural enhancement. Our work is independent of DNN architectures, which means that any advanced DNN can be utilized to minimize total computations.

Caching DNN outputs. Several studies [161, 156, 98] 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 [161] and ClassSR [156] use fewer computations on blocks that have lower difficulty, accelerating image super-resolution. These approaches can be applied to our solutions 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 [98] and FAST [218] apply super-resolution only to key frames, but this can greatly degrade quality on videos that contain dynamic scenes [210]. In contrast, we judiciously select a set of anchor frames given a computing budget in NEMO, considering their impact on the remaining non-anchor frames.

Caching DNN feature maps. DeepCache [208], DeepMon [143] and CBinfer [107] 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 §4.2.1). Based on this observation, our work 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, we utilize fine-grained frame dependencies embedded in a compressed video which have higher (quarter-pixel) precision in NEMO.

Optimizing model serving. Several efforts have been devoted to DNN-based model serving; they are orthogonal to our contributions. Clipper [114] proposes a layered architecture to ease the deployment of model serving on various ML frameworks and devices. Nexus [190], Clockwork [127], and InFaaS [186] 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 our ingest framework, NeuroScaler, to co-locate various types of streams in the same cluster.

Chapter 3. Neural-enhanced Video Streaming

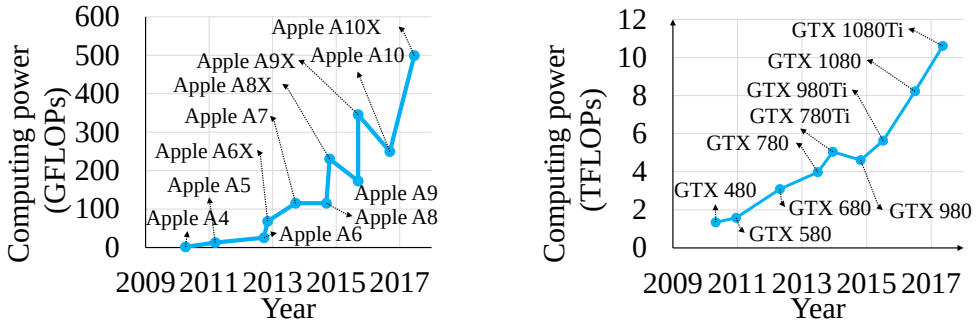
In this chapter, we present neural-enhanced video streaming that applies a deep neural network (DNN)-based quality enhancement on video *content* utilizing the *client computation* to maximize user quality of experience (QoE). This enables clients to obtain high-definition (e.g., 1080p) video from lower quality transmissions, providing a powerful mechanism for QoE maximization on top of bitrate adaption [211]. However, leveraging client computation via DNNs impacts the server/client system and introduces a number of non-trivial challenges. First, the CDN servers have to provide a DNN model for the content they provide, but it is difficult to guarantee the test performance of DNN's predictions. It is especially unreliable for unseen/new content, presenting a significant barrier to deployment. Second, client devices are heterogeneous. Their computational power varies widely and may even exhibit temporal variation due to multiplexing. Nevertheless, DNN-based quality enhancement must occur at real-time to support online video streaming. Finally, the DNN-based quality enhancement has a cascading effect on ABR-based QoE optimization. The quality now depends on the availability of DNNs at the client in addition to the available bandwidth. Thus, existing ABR algorithms must reflect the changes.

To address these challenges, we present NAS [212], which is the first system to apply super-resolution DNNs over video con- tent in the context of adaptive streaming. To guarantee reliable quality enhancement powered by DNN, it takes a content-aware approach in which a DNN is trained for each content separately. The idea is to leverage the DNN's overfitting property and use the training accuracy to deliver predictable high performance, instead of relying on the unpredictable test accuracy. Next, to meet the real-time constraints on heterogeneous environments, we use multiple scalable DNNs that provide anytime prediction [105, 135]. Such DNN architectures can adaptively control their computational cost given resource budget. NAS clients choose a DNN (from multiple options) that best fits their resources and adapt to temporal variations in computing power at each time epoch. The scalable DNN also enables the use of a partially downloaded DNN, bringing an incremental benefit in downloading a DNN model. Finally, to reconcile the ABR-based QoE optimization and DNN-based quality enhancement, we devise a content enhancement-aware ABR algorithm for QoE optimization. To this end, we integrate our design into the state-of-the-art ABR algorithm [169] that uses reinforcement learning [195]. The algorithm decides when to download a DNN model and which video bitrate to use for each video chunk.

This chapter is organized as follows. Section §3.1 provides the limitation of traditional video streaming and the potential of neural-enhanced video streaming. Section §3.2 identifies key challenges in enabling neural-enhanced video streaming and outlines the key design choices to solve these challenges. Section §3.3 and Section §3.4 describes the detailed design and implementation of NAS, respectively. Section §3.5 presents large-scale trace-driven evaluation that illustrates significant quality improvement by NAS. Finally, Section §3.6 concludes the chapter.

3.1 Motivation & Goal

Traditional approaches to improving video stream quality include: using better codecs [73, 74]; optimizing adaptive bitrate algorithms [96, 141, 147]; choosing better servers and CDNs [93, 166, 205]; and using coordination among clients and servers through a centralized control plane [167, 175]. These approaches focus on how to best utilize the network resource, but suffer from two common limitations.



(a) Mobile

(b) Desktop

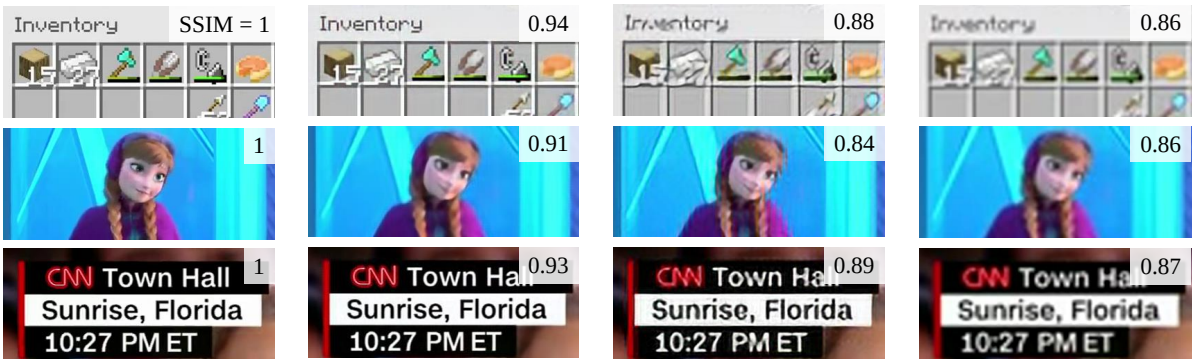
Figure 3.1: Growth of GPU's processing power

Under-utilization of client's computation. Market reports [66, 180] indicate the majority of users watch video primarily on PCs, which have significant computation power. Mobile devices, which is the next popular platform, are also equipped with power-efficient graphic processing units (GPUs) [120]. Figure 3.1 shows the exponential growth in GPU's computing power over time on mobile devices and desktop PCs. Latest mobile devices even have dedicated hardware for neural processing [38]. However, the current video delivery infrastructure *under-utilizes* client's computational power. With their growing computational capacity and ever-increasing demand for bandwidth, we envision a video delivery system in which clients take an active role in improving the video quality.

Limitation of current video coding. Video episodes often contain redundancy that occurs at large timescales. For example, consider a popular sports game (e.g., NBA finals) watched by millions of people. Same objects (e.g., balls and players) and scenes (e.g., basketball court) show up repeatedly. Similarly, redundancy is also found within episodes of a TV show, games in a sports league, and videos from the same streamers. Such frequently reoccurring high-level features contain valuable information that can be leveraged for video coding. However, standard video coding, such as MPEG and H.26x, only captures spacial and short-term redundancy, lacking any mechanisms to exploit motion picture's high-level features. Within a group of pictures (GOP), inter-frame coding encodes the difference between adjacent frames to compress a motion picture [121]. However, a GOP is typically on the order of seconds for online video [87], making it impossible to capture redundancy that occurs at large timescales. As long as codecs compress video only within a GOP (arguably a fundamental constraint for streaming), using sophisticated codecs would not completely close this gap.

Motivated by this, we envision a video delivery system that exploits such redundancy by capturing the high-level features and applies additional client computation to augment the limitation of traditional video encoding. To this end, we utilize DNNs that abstract meaningful features from a low-level representation of data [103].

System goal. Our goal is to design a practical system that augments the existing infrastructure to optimize user QoE. As the first step, we consider servicing on-demand videos, as opposed to live streams, and using personal computers that have desktop-class GPUs. We propose a redesign of the video delivery infrastructure to take advantage of client computation to a greater degree. For quality enhancement, we utilize super-resolution that takes low-quality video as input and generates an "up-scaled" version. We choose super-resolution because significant advances have been made recently [118, 154, 163]. While we scope our study to desktop-class GPUs and super-resolution, we believe the framework is generic enough



(a) Original (1080p) (b) Content-aware DNN (c) Content-agnostic DNN (d) 240p

Figure 3.2: 240p to 1080p super-resolution results
(Content type – 1st row: Game [90], 2nd row: Entertainment [89], 3rd row: News [92])

to accommodate different types of DNN models and devices.

3.2 Key Design Choices

Achieving our goal requires redesigning major components of video delivery. This section describes the key design choices we make to overcome practical challenges.

3.2.1 Content-aware DNN

Key challenge. Developing a universal DNN model that works well across all Internet video is impractical because the number of video episodes is almost infinite. A single DNN of finite capacity, in principle, may not be expressive enough to capture all of them. Note, a fundamental trade-off exists between generalization and specialization for any machine learning approach (i.e., as the model coverage becomes larger, its performance degrades), which is referred to as the 'no free lunch' theorem [206]. Even worse, one can generate 'adversarial' new videos of arbitrarily low quality, given any existing DNN model [140, 182], making the service vulnerable to reduction of quality attacks.

NAS' content-aware model. To tackle the challenge, we consider a content-aware DNN model in which we use a different DNN for each video episode. This is attractive because DNNs typically achieve near-zero training error, but the testing error is often much higher (i.e., over-fitting occurs) [194]. Although the deep learning community has made extensive efforts to reduce the gap [194, 144], relying on the DNN's testing accuracy may result in unpredictable performance [140, 182]. NAS exploits DNN's inherent overfitting property to guarantee reliable and superior performance.

Figure 3.2 shows the super-resolution results of our content-aware DNN and a content-agnostic DNN trained on standard benchmark images (NTIRE 2017 dataset [94]). We use 240p images as input (d) to the super-resolution DNNs to produce output (b) or (c). The images are snapshots of video clips from YouTube. The generic, universal model fails to achieve high quality consistently over a variety of contents—in certain cases, the quality degrades after processing.

In §3.3.1, we show how to design a content-aware DNN for adaptive streaming.

The content-aware approach can be seen as a type of video compression as illustrated in Figure 1.2.

Model name	Compute capacity (Single precision)	Price
GTX 1050 Ti	1.98 TFLOPS	$139
GTX 1060	3.86 TFLOPS	$249
GTX 1070	5.78 TFLOPS	$379
GTX 1070 Ti	7.82 TFLOPS	$449
GTX 1080	8.23 TFLOPS	$559
GTX 1080 Ti	10.61 TFLOPS	$669
Titan Xp	10.79 TFLOPS	$1,200

Table 3.1: Nvidia's desktop GPU (Geforce 10 series)

The content-aware DNN captures redundancy that occurs at large time scales (e.g. multiple GOPs) and operates over the entire video. In contrast, the conventional codecs deals with redundancy within a frame or between frames within a GOP.

In NAS, we demonstrate the new encoding scheme using per-video super-resolution DNNs. However, we believe the content-aware approach can be applied to a series of videos (of similar content) and extended to work with different types of DNNs, such as frame interpolation [177], as discussed in our position paper [211].

3.2.2 Multiple, Scalable DNNs

Key challenge. The available capacity of computing changes across time and space because of heterogeneity of client devices, changes in workloads, and multiplexing. Table 3.1 shows even within the desktop-class GPUs the computational power varies up to 5.68 times. Nevertheless, real-time inference is required for online video streaming—the DNN inference has to be at least as fast as the playback rate. However, existing super-resolution DNNs [118, 154, 163] require a fixed amount of computing power and cannot adapt to time-varying capacity. Thus, using a single DNN either under-utilizes client's GPU or does not meet the real-time requirement.

NAS' multiple, scalable DNN design. To tackle the challenge, NAS offers multiple DNNs and let clients dynamically choose one that fits their resource. Similar to multiple bitrates that adaptive streaming offers, we provide a range of DNN options that differ in their inference time (or computational requirements). NAS servers provide multiple DNN specifications as part of the video manifest file. We provide a light-weight mechanism that does not require clients to download the DNNs for choosing the right DNN from available options.

However, using multiple DNNs introduces another challenge. Because the size of DNN grows proportional to its computation requirement, DNNs designed for high-end GPU devices can be very large (a few MBs). It can take a long time to download and utilize the DNN. To address the issue, we design a scalable DNN that enables a client to utilize a partially downloaded model in an incremental fashion. The scalable DNN consists of multiple bypass-able intermediate layers, enabling a partial DNN without the intermediate layers to generate the output as shown in Figure 3.3. In addition, the design naturally accommodates temporal variation in computational power due to multiplexing. When the computational resource is abundant, clients can use all layers, otherwise they can opportunistically bypass any number of intermediate layers, enabling *anytime prediction* [105, 135, 160]. Finally, the use of multiple scalable DNNs allows each device to benefit from partially downloaded DNNs and provides the same level of

temporal adaptation regardless of the device's computational power. §3.3.2 presents the details of scalable DNNs.

3.2.3 Integrated ABR

Key challenges. As NAS uses per-video DNN, a client must download a DNN from a server to benefit from DNN-based quality enhancement. However, DNN downloads also compete for the bandwidth with the video stream itself. As a result, aggressively downloading the DNN model may degrade user QoE. At the same time, a client may benefit from an early DNN download because it can receive the quality enhancement early on. Because there exists a conflict, a careful decision making as to when and how to download the DNN is critical.

NAS' bitrate adaptation integrates the decision to download a DNN with bitrate selection for QoE maximization. It considers three additional factors that impact user QoE: 1) To benefit from quality enhancement, a client-side DNN must be downloaded first; 2) a partially downloaded DNN improves quality in proportion to the amount downloaded; and 3) DNN chunk downloads compete for bandwidth with video chunk downloads.

To solve the non-trivial problem, we leverage reinforcement learning [169] and generate an ABR algorithm that integrates the decision to download a DNN model. For this, we divide the DNN model into fixed-size chunks and train an RL network that outputs a decision (i.e., whether to download a video or a DNN chunk) using as input the current state (e.g., throughput measurement, playback buffer occupancy, the number of remained video chunks) and its history. We train the RL network using a large training set consisting of real network traces [62, 185].

The decision to use RL brings a number of benefits: 1) it allows NAS to directly optimize for any target QoE metric, while accounting for the benefit of DNN-based quality enhancement; 2) RL balances multiple interacting factors, such as bandwidth variations, bandwidth sharing between video and DNN chunks, and quality enhancement of partial DNNs, in a way that optimizes the QoE; and 3) it naturally accommodates the use of multiple DNNs by encoding the DNN type in the RL network. §3.3.3 presents the details of the integrated ABR.

3.3 System Design

NAS is implemented on top of current HTTP adaptive streaming, standardized in DASH [17]. We start by explaining the key differences in how the system operates.

New video admission (server-side processing). As in DASH, when a video clip is uploaded, it is encoded at multiple bitrates and divided into chunks. In addition, the server trains content-aware DNNs for the video for client-side quality enhancement (§3.3.1). It then associates the DNNs with the video by placing their URLs in the manifest file along with DNN specifications (§3.3.2).

Client behavior. A client's video player first downloads a manifest file, which contains a list of available DNNs for the video. The client then selects one of them that fits its computing power. The client's DNN processor uses a light-weight mechanism to choose the best available one that fits its resource (§3.3.2). The player then downloads the selected DNN or video chunks following the decision given by the integrated adaptive bitrate (ABR) algorithm (§3.3.3). When a DNN chunk is downloaded, the player passes it to the DNN processor, and the processor loads the (partial) DNN on the client's computing device (e.g. GPU). When a video chunk is downloaded, the player keeps it in the playback buffer and the

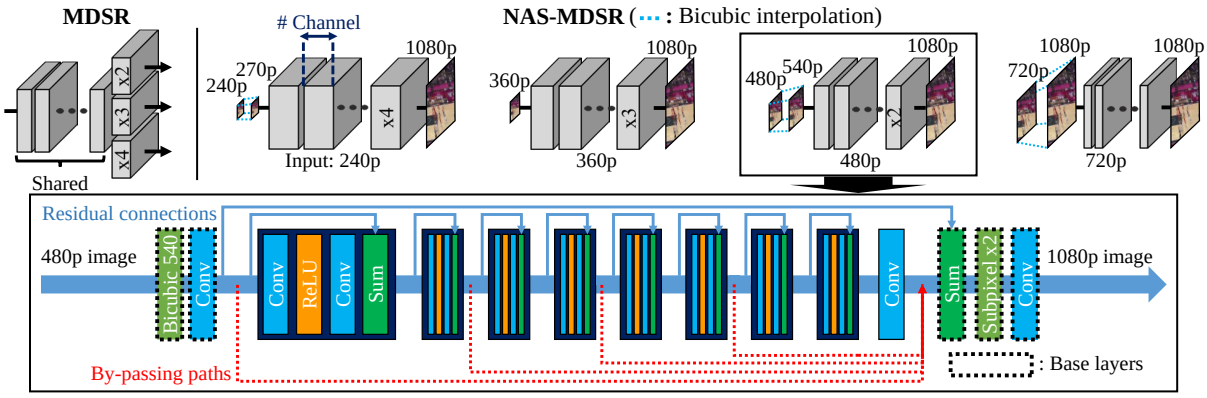


Figure 3.3: Super-resolution DNN architectures: MDSR vs. NAS-MDSR

chunk becomes ready for immediate playback. The player then opportunistically passes video chunks in the playback buffer to the DNN processor for quality enhancement along with their associated playback time, which indicates the deadline for neural processing. When the DNN processor finishes processing a chunk, it replaces the original chunk in the playback buffer. Finally, the quality enhanced video chunk is played.

Client-side neural processing. The DNN processor initializes the DNN as DNN chunks arrive. The DNN we use performs quality enhancement on a per-frame basis for super-resolution. Thus, the DNN processor first decodes a video chunk into frames. The DNN processor then applies the super resolution DNN. The resulting frames are then re-encoded to video chunks which replace the original chunks in the playback buffer. The decoding, super-resolution, and encoding phases are pipelined and parallelized to minimize the latency (See §3.5.5 for details).

3.3.1 Content-aware DNN for DASH

Applying DNN to adaptive streaming. Standard DNN architectures are not designed for adaptive streaming which introduces specific requirements. First, because adaptive streaming uses multiple resolutions, DNN must be able to take multiple resolutions as input. Second, the DNN inference has to take place in real-time. Finally, DNN should sacrifice its inference quality as little as possible in meeting the first two requirements. The inference time can be reduced at the cost of quality by reducing the number of layers and/or the number of channels (a set of features) in each layer. Such down-scaling also decreases DNN's network footprint. Thus, we must strike a balance between the tradeoff in quality and size, while meeting the real-time requirement.

Using a super-resolution network. Our system extends MDSR [163], a state-of-the-art super-resolution network. As shown in Figure 3.3, MDSR supports multi-scale super-resolution (x2, x3, x4) in a single network, while sharing the intermediate layers to reduce the footprint. The input resolution drastically affects the inference time of MDSR. For example, even on a flagship desktop GPU (e.g., Nvidia Titan Xp), a 720p input only delivers 3.23 frames per second, whereas a 240p image is processed nearly in real-time at 28.96 frames per second. Thus, to meet the real-time constraint for the highest resolution, one has to downscale the network size.

However, due to the shared layer design, this degrades the quality of all resolutions uniformly, making the lower resolution suffer from significant quality degradation. To avoid the limitation, we use a separate network for each resolution (Figure 3.3), trading in the total size of the DNN for inference quality. For

each DNN, we fix the number of layers that represents the capacity to adapt to temporal variation in computing power (§3.3.2). Then, we take the maximum number of channels, independently, for each resolution, that satisfies the real-time constraint. The resulting DNN configuration and size we use for our evaluation are listed in Table 3.2. The footprint of the 'Ultra-high' DNN is 2,145 KB, which is about the size of a single 1080p (four-second) video chunk (4.8 Mbps) from our evaluation. The size of the 'Low' DNN is only about half the size of a 240p chunk (400 Kbps). While we use NAS-MDSR and specific settings for evaluation, NAS design is not bound to any specific DNNs but accommodates their evolution.

Training content-aware DNNs. The training data is pairs of a low resolution (e.g., 240p, 360p, 480p, 720p) and the original highest resolution (e.g., 1080p) image. We update the DNN parameters to minimize the difference between the DNN's output and the target high resolution image. The training cost is of "one-time" and is amortized across the total watch time of the video episode. Nevertheless, for CDNs that deliver many video episodes, the total computational cost may be high. To reduce the cost of training, we apply the fine-tuning strategy of a transfer learning type. Namely, we first train a generic DNN model using a popular standard benchmark [94]. We then train the content-aware DNN model on each video episode with its weights initialized as those from the generic model. This reduces the computation cost in training by 5-6x, while achieving similar quality enhancement compared to random initialization [132].

3.3.2 Adaptation to Computational Power

This section describes two mechanisms (multiple DNNs and anytime prediction) that clients use to adapt to their computational capacity to deliver real-time quality enhancement. For each technique, we describe the enabling DNN design and explain the client-side dynamic adaptation logic.

Providing multiple DNNs (server-side). As discussed in §3.2.2, we provide multiple DNN configurations that vary in quality and computational requirements to support a broad range of GPUs. Similar to GPU's rendering quality options, we provide four quality levels of DNNs: 'Low', 'Medium', 'High', 'Ultra-high'. Thus, we have a DNN per quality per input-resolution, as shown in Table 3.2. Finally, the server records the available DNN configurations on a manifest file, including the DNN name and level, input resolution, the number of layers, and the number of channels.

Choosing a DNN from multiple options (client-side). Clients test-run the DNN options to choose the one that gives the best quality improvement and delivers real-time performance. A naive way to measure the inference time of DNNs is downloading all DNNs at the client device. However, this consumes large bandwidth (several MBs) and unnecessarily delays video streaming, ultimately degrading user QoE. To streamline the process, NAS provides enough information about the DNN options (i.e., the number of layers and channels) in the manifest file for clients to reconstruct mock DNNs without downloading the DNNs. Using the DNN configuration defined in the manifest file, clients generate DNNs initialized with random weights and run them on their GPUs. Finally, the clients select the largest (highest-quality) DNN that runs in real-time—the client does not need actual weights here because a larger DNN provides better quality. With four DNN options, the client-side test-run takes between 1.64-3.40 seconds depending on a GPU model. Thus, the client can decide which DNN to use early on without downloading any DNN.

Scalable DNN and anytime prediction (server-side). Our scalable DNN architecture enables the client to utilize a partially downloaded DNN and adapt to time-varying computational power. Utilizing partial DNNs provides incremental benefit as the download progresses. This especially benefits the QoE at the beginning of a video because the full DNN of a few MBs cannot be transferred instantly. In addition, the scalable architecture enables anytime prediction allowing us to adapt to client's available computational power that may change unexpectedly.

Input Resolution	DNN Quality Level			
	Low	Medium	High	Ultra-high
240p	20, 9 43 KB	20, 21 203 KB	20, 32 461 KB	20, 48 1026 KB
360p	20, 8 36 KB	20, 18 157 KB	20, 29 395 KB	20, 42 819 KB
480p	20, 4 12 KB	20, 9 37 KB	20, 18 128 KB	20, 26 259 KB
720p	6, 2 2 KB	6, 7 5 KB	6, 16 17 KB	6, 26 41 KB

Table 3.2: DNN configurations for NAS-MDSR
(#Layer, #Channel, Size)

For this, we modify the DNN architecture and its training method. The DNN's intermediate layers consist of multiple residual blocks [163] each of which consists of two convolutional layers. We allow bypassing consecutive intermediate blocks during inference. To enable bypassing, we add direct connections from intermediate blocks to the final layers, as shown in Figure 3.3. This creates multiple inference paths as shown in the figure. We then train all interference paths in the following way.

In each training iteration, we randomly bypass intermediate layers to calculate the error between the network output and the target image. Then, we use back-propagation [187] to update parameters. In particular, we go through all layers with probability 1/2 (for training the original path) and choose one of the remaining by-passing paths uniformly at random otherwise. The resulting DNN can generate an output image using only a part of the DNN and provide incremental quality improvement as more layers are used. Finally, DNNs are divided into chunks. Our server places the chunks' URLs in the video manifest file. The first DNN chunk consists of the base layers for all input resolutions. The subsequent chunks contain the rest of DNNs.

Using the scalable DNN (client-side). A client downloads the DNN in chunks. It reconstructs the first *partial* DNN after downloading the base layers. The size of this minimal DNN is only 35.11% – 36.14% of the full DNN (33 KB to 768 KB), allowing the client to start benefiting from DNN-based quality enhancement early on. As the client downloads more blocks, it updates the DNN, which provides incremental benefit.

Finally, our client opportunistically determines the number of layers to use during video playback. At every time interval, the client's DNN processor first calculates the amount of time remaining until the playback time of the chunk it is processing. The client then calculates the maximum amount of layers it can use that meets the deadline. To aid this, the client records the latest inference time for each layer and updates this table when the inference time changes. We empirically set the time interval to four seconds, which is the length of a single video chunk in our evaluation. This allows NAS clients to dynamically adapt to changes in the available computational power, as we demonstrate in §3.5.4.

3.3.3 Integrated Bitrate Adaptation

NAS integrates two decisions into its ABR algorithm for QoE optimization: 1) it decides whether to fetch a video chunk or a DNN chunk; and 2) if the first decision is to fetch a video chunk, it chooses the

Type	State
DNN status	# of remaining DNN chunks
Network status	Throughput for past N chunks
	Download time past N chunks
Player status	Playback buffer occupancy
	Next video chunk sizes
Video status	Bitrate of the latest video chunk
	# of remaining video chunks

Table 3.3: State used in our RL framework. We use $N = 8$ which empirically provides good performance.

chunk's bitrate.

The algorithm must balance the two conflicting strategies. The first strategy places emphasis on downloading the DNN model in the hope that this will bring quality enhancement in the future, while sacrificing video's streaming quality at the moment. The second strategy optimizes for video bitrate at the moment and delays the DNN download. In practice, the resulting outcome is unpredictable because it depends on how the network conditions change. The solution space is extremely large considering the number of bitrates, the download order of video and DNN chunks, and the dynamic range of available bandwidth.

To tackle the challenge, we use a reinforcement learning (RL) framework [169, 174] that directly optimizes the target metric (without using explicit decision labels) through comprehensive "experience". In particular, we adopt the actor-critic framework of A3C [174]. It learns a strategy (or policy) from observations and produces a mapping from raw observations, such as the fraction of DNN model downloaded, the quality improvement due to DNN, network throughput samples, and playback buffer occupancy, to the decisions described above.

RL design. An RL agent interacts with an environment [195]. For each iteration t, the agent takes an action a_t, after observing a state s_t from the environment. The environment then produces a reward r_t and updates its state to s_{t+1}. A policy is defined as a function that gives the probability of taking action a_t given s_t, $\pi(s_t, a_t) :\rightarrow [0, 1]$. The goal then is to learn a policy, π, that maximizes the sum of future discounted reward $\sum_{t=0}^{\infty} \gamma^t r_t$, where $\gamma \in (0, 1]$ is a discount-rate for future reward.

In our case, the set of actions $\{a_t\}$ includes whether to download a DNN chunk or to download a video chunk of a specific bitrate. The state s_t includes the number of remaining DNN chunks to download, throughput measurements, and player measurements (e.g., the playback buffer occupancy, past bitrates). Table 3.3 summarizes the state s_t. The reward r_t is the target QoE metric which is a function of bitrate utility, rebuffering time, and smoothness of selected bitrates [214, 169] defined as:

$$\frac{\sum_{n=1}^{N} q(R_n) - \mu \sum_{n=1}^{N} T_n - \sum_{n=1}^{N-1} \left| q(R_{n+1}) - q(R_n) \right|}{N} \tag{3.1}$$

where N is the number of video chunks; R_n and T_n respectively represent the video chunk n's bitrate and the rebuffering time resulting from its download; μ is the rebuffering penalty; and $q(R_n)$ is the perceived quality of bitrate R_n (refer to Table 3.5 in §3.5.1 for the choices of μ and $q(R_t)$).

Component	Lines of code (LoC)	Changed
DASH video player	19K lines of JavaScript	8.8% (1763)
Content-aware DNN	6.3K lines of Python	- (6.3K)
Integrated ABR algorithm	5.5K lines of Python	- (5.5K)

Table 3.4: NAS implementation (Lines of Code)

To reflect the DNN-based quality enhancement of NAS, we define effective bitrate $R_{\texttt{effective}}$ instead of the nominal bitrate R_n. For each video chunk C_n:

$$R_{\texttt{effective}}(C_n) = \text{SSIM}^{-1}(\text{SSIM}(\text{DNN}_m(C_n)))$$

where $\text{DNN}_m(C_n)$ represents the quality enhanced video chunk C_n after downloading the (partial) DNN chunk m, SSIM is the average structural similarity [203] for measuring the video quality, and its inverse SSIM^{-1} maps a SSIM value back to the video bitrate. To create the mapping, we measure the SSIM of original video chunks at each bitrate (or resolution) and use piece-wise linear interpolation (e.g., (400 Kbps, SSIM_1), ..., (4800 Kbps, SSIM_5)).

RL training. Our RL framework has two neural approximators: an actor representing the policy and a critic used to assess the performance of the policy. We use the *policy gradient method* [196] to train the actor and critic networks. The agent first generates trajectories following the current policy $\pi_\theta(s_t, a_t)$, where θ represents parameters (or weights) of the actor's neural network. The critic network observes these trajectories and learns to estimate the *action-value function* $Q^{\pi_\theta}(s_t, a_t)$, which is the total expected reward with respect to taking action a_t starting at state s_t and following the policy π_θ. At each iteration, the actor network uses this estimation to update the model:

$$\theta \leftarrow \theta + \alpha \sum_t \nabla_\theta \log \pi_\theta(s_t, a_t)(Q^{\pi_\theta}(s_t, a_t) - V^{\pi_\theta}(s_t)),$$

where $V^{\pi_\theta}(s_t)$ is the *value function* representing the total expected reward of π_θ starting at state s_t, and α is the learning rate. In our RL framework, because the reward reflects the average QoE enhancement that content-aware DNN delivers, the critic network learns to estimate the updated total reward. This enables the actor network to learn a policy that balances video and DNN downloads to maximize the QoE.

We use a chunk-level simulator similar to that of Pensieve to accelerate the ABR training. It takes network throughput traces and simulates NAS' video streaming dynamics. In addition, we pre-compute the DNN-based quality enhancement by averaging it over all the videos for each DNN quality. We then use the values to produce a generic ABR model.

When a DNN is downloaded, the simulator updates the amount of downloaded DNN chunks (i.e., decrements the state 'number of remaining DNN chunks'). When a video chunk is downloaded, it adds the chunk to the playback buffer. It then computes the QoE that reflects DNN-based quality enhancement, using the (effective) bitrate utility of each chunk and the rebuffing time. Note, the simulator performs neither actual video downloads nor DNN inferences. Thus, it reduces the training time by 97.12% compared to real-time emulation.

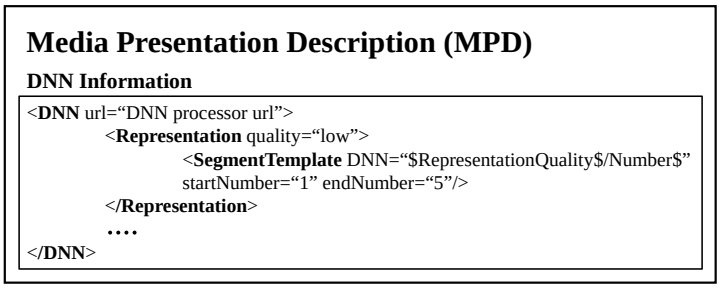


Figure 3.4: NAS manifest file structure

3.4 Implementation

We implement NAS client by extending a DASH video player. Both the server-side (training) and client-side (inference) DNN processing are implemented using Pytorch [183]. Table 3.4 shows the lines of code (LoC) for each component.

NAS client (DASH video player). To implement NAS client, we modify dash.js [19] (version 2.4), a reference implementation of MPEG DASH client written in JavaScript. We run the integrated ABR and content-aware DNNs as separate processes. dash.js is configured to fetch the ABR decisions and quality enhanced chunks through inter-process communication. We add DNN metadata on a manifest file as shown in Figure 3.4. The `quality` attribute indicates the DNN quality level. The `DNN` attribute of `SegmentTemplate` is used to create the chunk URL, and `startNumber` and `endNumber` indicate the chunk index range. In addition, the manifest file includes the number of layers and the number of channels for each DNN.

Training content-aware DNNs. We implement the scalable super-resolution DNN using Pytorch. For training the DNN model, we use input image patches of size 41x41 pixels by randomly cropping the low-resolution images (e.g., 240p, 360p, 480p, 720p) and run the popular ADAM algorithm [155] to optimize DNN parameters. The mini-batch size, weight decaying parameter, and learning rate are set to 64, 10^{-3}, and 10^{-4}, respectively. We initialize the DNN model using parameters of the generic model (§3.2.1). We then fine-tune it over 100 mini-batch updates per minute of video to generate a content-aware DNN. Finally, we round off its parameters from single-precision (32-bit) to half-precision (16-bit), which halves the DNN size while introducing minimal performance degradation (virtually no difference in SSIM).

Training integrated ABR. We implement our integrated ABR extending Pensieve's implementation [57]. The initial learning rates of actor/critic networks are set to 10^{-4} and 10^{-3}, respectively. The entropy weight is initialized as 2. We iterate training over 60,000 epochs in which we decay the entropy weight from 2 to 0.055 exponentially over every epoch. Finally, we select the ABR network that delivers the highest QoE for our training traces.

3.5 Evaluation

We evaluate NAS by answering the following questions.

- How does NAS perform compared to its baseline competitors, and what is the training cost?
- How does each design component of NAS contribute to the overall performance?

QoE type	Bitrate utility ($q(R)$)	Rebuffer penalty (μ)
$\text{QoE}_{\texttt{lin}}$	R	4.3
$\text{QoE}_{\texttt{log}}$	$log(R/R_{min})$	2.66
$\text{QoE}_{\texttt{hd}}$	0.4→1, 0.8→2, 1.2→3 2.4→12, 4.8→15	8

Table 3.5: QoE metrics used for evaluation

- Does NAS effectively adapt to heterogeneous devices and temporal variance in client's computing power?
- What is the end-to-end processing latency and resource usage of NAS?

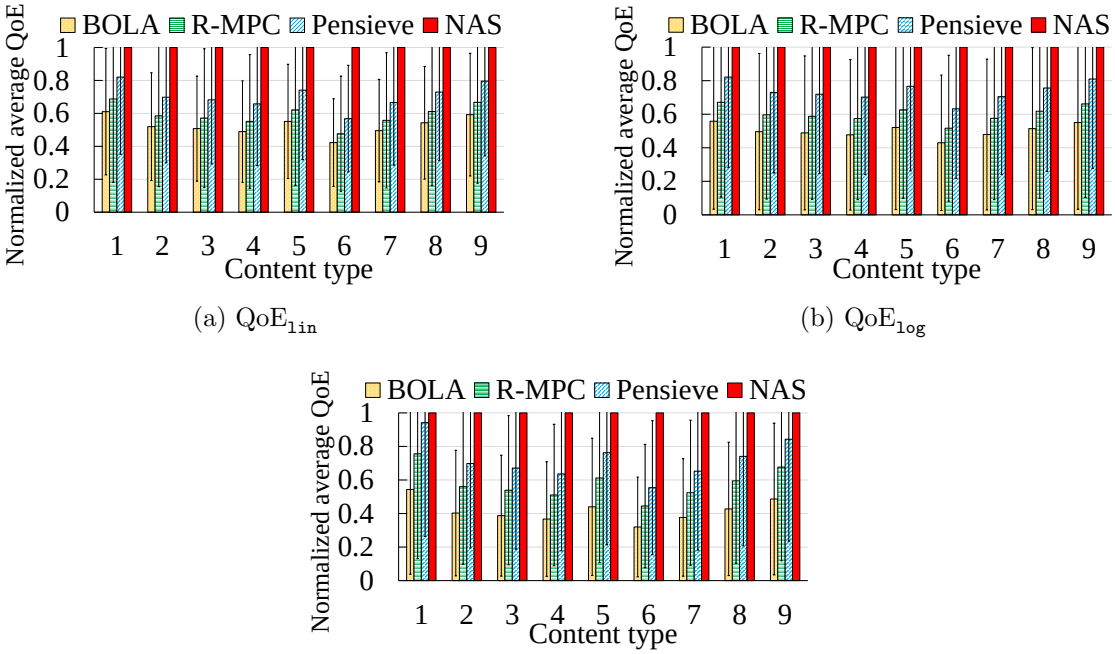
3.5.1 Methodology

Videos. We use videos from popular channels on Youtube. For each of the nine Youtube channel categories, we select three popular channels in the order of appearance. We then pick the most popular video from each channel that supports 1080p quality and whose length is longer than 5 minutes—the number of views of 27 video clips ranges from 7M to 737M. Finally, we download 1080p videos and produce multi-bitrate videos following the Youtube and DASH recommendations [87, 18]: Each 1080p video is re-encoded using the H.264 codec [73] in which GOP (or chunk size), frame rate, and bitrates are respectively set to 4 seconds, 24 fps and {400, 800, 1200, 2400, 4800}Kbps (which represent for {240, 360, 480, 720, 1080}p resolution videos). Unless otherwise noted, we use the entire video for training and use the first 5 minutes for playback. Training over the entire video ensures that NAS delivers consistent quality enhancement over the entire video.

Network traces. We use a real bandwidth dataset consisting of 508 throughput traces from Norway's 3G network (2010-2011) [185] and 421 traces from U.S. broadband (2016) [62], compiled by the Pensieve author [57]. We filter out the traces that consistently experience low bandwidth (< 400 Kbps) for an extended time ($\geq$100 seconds). The resulting average throughput ranges from 0.38 Mbps to 4.69 Mbp, and the mean and median are 1.31 Mbps and 1.09 Mbps, respectively. Each trace spans 320 seconds, and we loop the trace until a video is completely downloaded. We use randomly selected 80 % of our traces for training and the remaining 20 % for testing.

Baseline. We compare NAS against the following state-of-the-art bitrate adaptation that does not utilize client computation.

- Pensieve [169] uses deep reinforcement learning to maximize QoE.
- RobustMPC [214] uses playback buffer occupancy and throughput predictions over next five chunks to select the bitrate that maximizes QoE. We use the version reproduced by the authors of Pensieve [57].
- BOLA [193] uses Lyapunov optimization based on playback buffer occupancy. We use the BOLA version implemented in dash.js, which is a Javascript-based reference implementation of a MPEG-DASH player [19].



(a) $\text{QoE}_{\texttt{lin}}$ (b) $\text{QoE}_{\texttt{log}}$

(c) $\text{QoE}_{\texttt{hd}}$

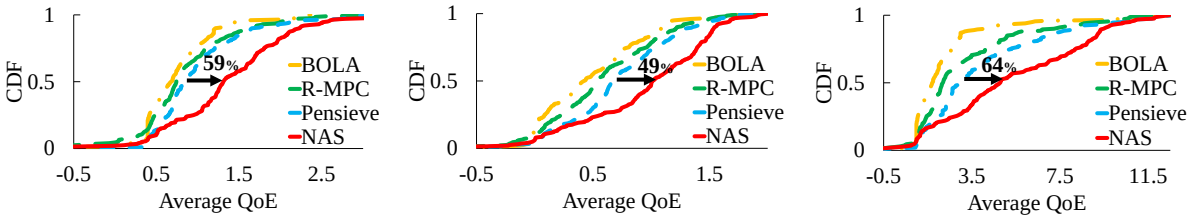
Figure 3.5: Normalized QoE comparison of video clips from the nine content categories of YouTube. (1: Beauty, 2: Comedy, 3: Cook, 4: Entertainment, 5: Game, 6: Music, 7: News, 8: Sports, 9: Technology)

QoE metrics. We use three types QoE metrics, compiled by MPC and Pensieve, whose the bitrate utility function, $q(R_n)$, and rebuffering penalty constant, μ of Equation 3.1, differ as summarized in Table 3.5.

- $\text{QoE}_{\texttt{lin}}$ uses a linear bitrate utility.
- $\text{QoE}_{\texttt{log}}$ uses a logarithmic bitrate utility function that represents its decreasing marginal utility.
- $\text{QoE}_{\texttt{hd}}$ heavily favors high-definition (HD) video (720p and 1080p) over non-HD.

Experimental settings. We run our dash.js implementation on a Chromium Browser (version 65) to stream MPEG-DASH videos. We use six GPU models from Nvidia's desktop GPU product line listed in Table 3.1. Unless otherwise noted, the default client-side GPU is Nvidia Titan Xp. In our setting, the content-aware DNN and the ABR network run at the client as separate processes. To emulate the network conditions from the network traces, we use Mahimahi [176].

We use two experiment settings. To evaluate NAS client on all six GPUs, we have a local testbed. To scale training and testing, we use Google Cloud Platform. Training is done using GPU instances equipped with Nvidia's server-class Tesla P100 GPU. However, Google Cloud Platform does not have desktop class GPUs, while we need to scale client-side streaming experiments to 18 hours of network traces x 27 video clips x 4 types of ABR x 3 types of QoE, totaling 5,832 hours of streaming time. Thus, we take quality and latency measurements of content-aware DNNs using the local testbed on each GPU device for each video. We then emulate the network condition between NAS server and client once for each network trace and apply the effect of the quality enhancement and latency of content-aware DNNs.



(a) QoE_{lin} (b) QoE_{log} (c) QoE_{hd}

Figure 3.6: Cumulative distribution of QoE for 'Sports' content category

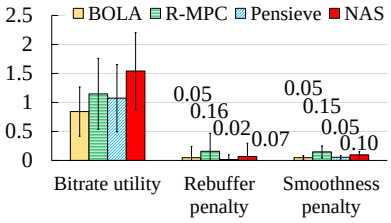


Figure 3.7: QoE_{lin} breakdown

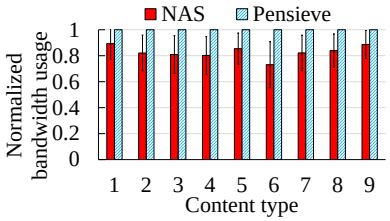


Figure 3.8: Normalized bandwidth usage at QoE_{lin}=0.98

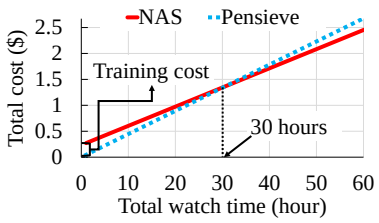


Figure 3.9: Cumulative server cost

We confirm the network-emulated, DNN-simulated clients produce the same QoE as real clients of our local testbed using a fraction of test data.

3.5.2 NAS vs. Existing Video Delivery

QoE improvement. Figure 3.5 shows the average QoE of video clips across the nine content categories. The error bars indicate one standard deviation from the average. NAS delivers the highest QoE across all content categories over all three QoE metrics. The result shows significant improvement over prior work. NAS consistently outperforms Pensieve by a large margin across all QoE metrics: QoE_{lin} (43.08% better), QoE_{log} (36.26% better), and QoE_{hd} (42.57% better). With QoE_{lin}, NAS outperforms Pensieve 43.08% on average, whereas Pensieve achieves a 19.31% improvement over RobustMPC (R-MPC). Compared to BOLA, NAS achieves 92.28% improvement in QoE_{lin}. The QoE improvement varies across content types from 21.89% (1: 'Beauty') to 76.04% (6: 'Music') over Pensieve because many factors, such as the scene complexity, compression artifacts, and temporal redundancy, affect the DNN performance.

Figure 3.6 shows the cumulative distribution of QoE over our test traces. It shows the 'Sports' content category which shows medium gain among all categories. NAS delivers benefit across all network conditions. NAS improves the median QoE_{lin} by 58.55% over Pensieve. Note, Pensieve mainly delivers its QoE gain over RobustMPC by reducing rebuffering at the cost of bitrate utility. In contrast, NAS does not exhibit such tradeoff because it uses client computation. Other content (not shown) displays a similar trend. Finally, Figure 3.7 shows a breakdown of QoE into bitrate utility, rebuffering penalty, and the smoothness penalty. NAS benefits the most from the bitrate utility due to the DNN-based quality enhancement.

Bandwidth savings. Despite the DNN transfer overhead, NAS requires less bandwidth in delivering the same QoE level. To demonstrate this, we create a hypothetical setting using the chunk-level simulator (§3.3.3) where NAS clients receive a fraction of bandwidth that Pensieve clients receive including the DNN

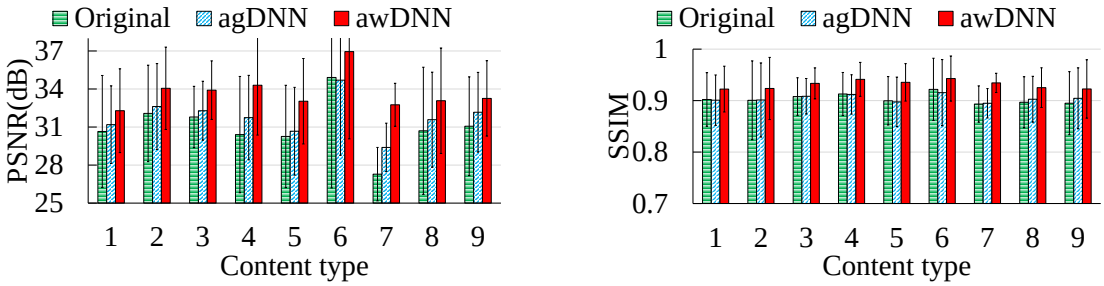


Figure 3.10: Video quality in PSNR and SSIM (240p input)

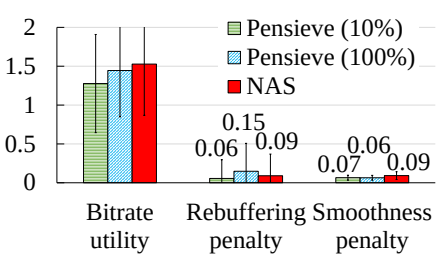
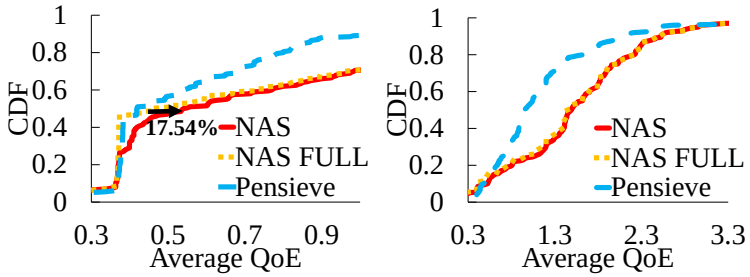


Figure 3.11: Integrated ABR vs. Baseline algorithm



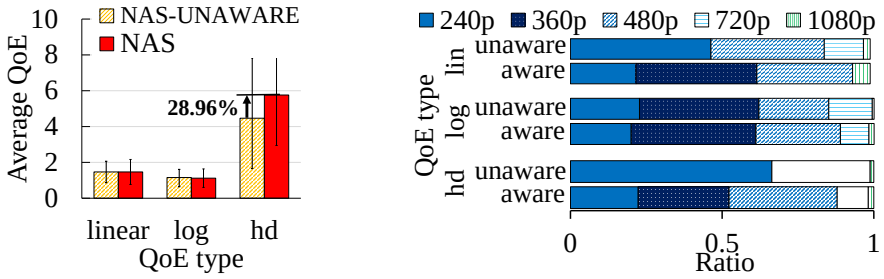
(a) t $\leq$ 47.82 seconds (b) t $>$ 47.82 seconds

Figure 3.12: Scalable DNN vs. Full DNN

transfer overhead. We adjust the fraction and empirically determine the fraction that delivers the same QoE. We assume NAS clients download the largest DNN ('Ultra-high') model for every five-minute video. Figure 3.8 shows the average bandwidth usage of Pensieve and NAS. On average across all videos, NAS requires 17.13% less bandwidth than Pensieve to deliver the same quality. The savings vary across content between 10.69% and 26.90%. This demonstrates the benefit of using a DNN outweighs the overhead of transferring the DNN.

Cost-benefit analysis (server-side). We now quantify the overall server-side cost in using a NAS content delivery network. While NAS servers use less bandwidth to deliver the same quality, they must train content-aware DNNs and the integrated ABR network. We quantify the computation and bandwidth cost of the CDN servers. The training time for the integrated ABR is only 10.92 hours on a CPU. Because it is a one-time cost amortized across all video streams, the additional cost is negligible. In contrast, the content-aware DNNs must be trained for each video. The total training time (across multiple DNNs) per minute of video is 10 minutes.

For a Google cloud instance with 8 vCPUs, 32 GB RAM, and a Nvidia P100 GPU, this translates to \$0.23 per minute of video. For bandwidth, Amazon CDN instance charges at most 0.085 \$/GB. The price per bandwidth becomes cheaper as one uses more bandwidth. Using these as reference, we compute the total video delivery cost a function of cumulative viewing time per minute of video. Figure 3.9 shows the cost comparison for NAS and Pensieve. As before, we assume each user watches a video clip for five minutes (i.e., DNNs are transferred every five minutes of viewing). This is a conservative estimate given the popularity of binge-watching [15]. NAS pays the up-front cost of computation, but as the cumulative viewing time increases, it is amortized. Note, NAS uses 17.13% less bandwidth to deliver the same user QoE. Thus, when the cumulative viewing reaches 30 hours (per minute of video in the system), NAS CDN recoups the initial investment.



(a) Average QoE

(b) Video quality distribution

Figure 3.13: Integrated ABR vs. Quality-unaware ABR

3.5.3 Component-wise Analysis

We evaluate how each design component contributes to the quality improvement.

Content-awareness. Figure 3.10 compares video quality of content-aware DNN (awDNN), a content-agnostic DNN (agDNN) trained on standard benchmark images (NTIRE 2017 dataset [94]), and the original 240p video we use as input upscaled by the bicubic interpolation. We measure the video quality both in PSNR [133] and SSIM [203] in which PSNR represents the average mean square error between two images in logarithmic decibel scale. Content-aware DNN delivers consistent improvement whereas content-agnostic DNNs even degrades the quality in some cases with respect to the PNSR measure (content type: 6) and the SSIM measure (type: 1,2,4,5,6). This confirms our rationale for using DNN's training accuracy.

Scalable DNN. Figure 3.12 demonstrates the benefit of utilizing a partial DNN. We compare Pensieve, NAS, and a version of NAS (`NAS-FULL`) that does not utilize partial DNN downloads. Specifically, Figure 3.12(a) shows the cumulative distribution of $\text{QoE}_{\texttt{lin}}$ before the average full DNN download time (47.82 seconds). As soon as a partial DNN is downloaded (22.16 seconds on average), NAS enhances the quality. The result shows that this delivers 17.54 % and 3.35 % QoE improvement in the median and mean, respectively. Note, the QoE of NAS and `NAS-FULL` becomes identical after downloading the full DNN ($t > 47.82$ seconds) as shown in Figure 3.12(b).

Integrated ABR. The integrated ABR delivers benefit during and after the DNN download. To demonstrate this, we create two hypothetical settings using the chunk-level simulator (§3.3.3).

First, we compare NAS with a version that uses a naive DNN download strategy that downloads a DNN at a fraction of the video bitrate chosen by Pensieve. Note, it does not integrate the bitrate selection and DNN download decision. We use two variants: one that aggressively downloads DNN at 100% of the video bitrate and the other that uses only 10%. Both are configured to start downloading the DNN when the playback buffer becomes larger than 15.42 seconds, which is the average time that NAS starts to stream a DNN in our test traffic traces. Our result shows NAS respectively outperforms the non-aggressive and aggressive strawman by 16.36% and 9.13% with respect to $\text{QoE}_{\texttt{lin}}$. Figure 3.11 shows the comparison of QoE components. The non-aggressive version experiences lower bitrate utility compared to NAS because the former downloads the DNN more slowly. In contrast, the aggressive version increases the rebuffering penalty by x2.5 which negatively affects the QoE.

Next, to evaluate the benefit of quality-enhancement aware bitrate selection after the DNN is fully downloaded, we compare NAS with a quality-enhancement unaware ABR after the full DNN download. Figure 3.13(a) shows the average QoE in this setting. We see that the quality-enhancement aware ABR

Model Name	Frames per second (FPS)			
	Low	Medium	High	Ultra-high
GTX 1050 Ti	**34.36**	17.62	14.91	7.37
GTX 1060	45.27	**30.05**	25.96	13.17
GTX 1070 Ti	41.76	45.24	**41.53**	21.47
GTX 1080	53.82	52.86	**38.95**	21.46
GTX 1080 Ti	58.94	56.29	57.12	**31.34**
Titan Xp	52.99	51.72	52.22	**33.58**

Table 3.6: DNN processing speed on desktop GPUs (Bold font indicates the selected quality level.)

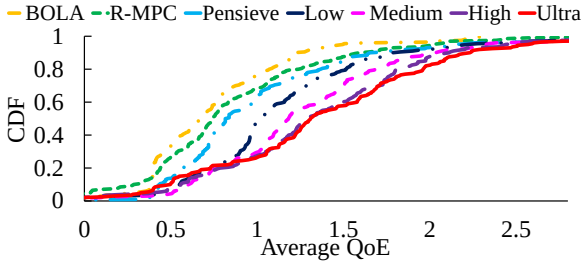


Figure 3.14: CDF of QoE over different quality DNNs

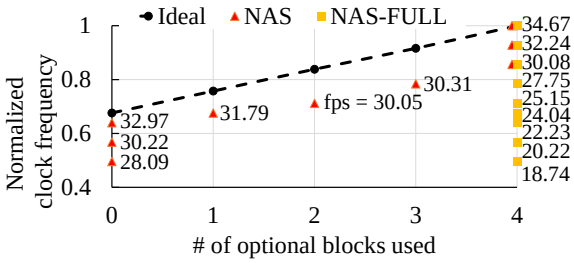


Figure 3.15: Dynamic adaptation to computing power

delivers a large gain for $\text{QoE}_{\texttt{hd}}$ (28.96 %), whereas it offers minimal benefit to $\text{QoE}_{\texttt{lin}}$ (0.01 %) and slightly degrades on $\text{QoE}_{\texttt{log}}$ (-3.14 %). The reason it delivers a large gain for $\text{QoE}_{\texttt{hd}}$ is because the DNN-enhanced quality of 1.2 Mbps (480p) videos get close to that of the original 2.4 Mbps (720p) video and the marginal utility with respect to the increased quality is far greater for $\text{QoE}_{\texttt{hd}}$ than any other QoE type, especially between 1.2 Mbps to 2.4 Mbps (Table 3.5). The integrated ABR reflects this enhancement in the bitrate decisions to download 480p much more often when using $\text{QoE}_{\texttt{hd}}$ as shown in Figure 3.13(b).

3.5.4 Dynamic Adaptation to Computation

We demonstrate NAS's ability to adapt to heterogeneous clients and temporal variation in computing power.

Heterogeneous clients. We demonstrate NAS is able to meet real-time constraints on six desktop GPUs shown in Table 3.1. We run our DASH client on six clients each with a different GPU model and measure their performance. First, we measure the throughput of the DNN processing engine, which includes decoding, DNN inference, and re-encoding. Table 3.6 reports the minimum processing speed across all input resolutions for each device. The video playback rate is 30 frames per second. Clients perform a test-run when it receives a video manifest file. The selected quality level (e.g., 'Low', 'Medium', 'High', or 'Ultra-high') is indicated in boldface. We see that each device selects one that meets the real-time constraint. Note the processing time does not depend on video content.

Next, we measure the QoE of clients using four different quality levels in our cloud setting. Figure 3.14 shows the cumulative distribution of $\text{QoE}_{\texttt{lin}}$ for each quality level. All quality levels outperform Pensieve. The higher the quality level DNN, the better the quality it delivers. Note, even though DNNs of higher quality are larger in size, they deliver incremental benefit over lower quality DNNs.

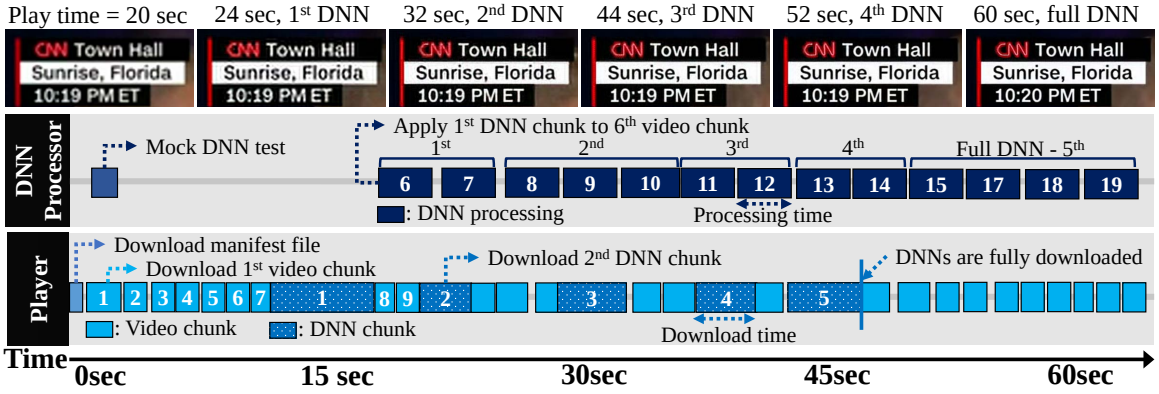


Figure 3.16: Case study: A time-line of NAS client in operation (Video Source: [92])

Phase	Processing time (sec)
Decode	0.28
Super resolution	3.69
Encode	0.91
Total	4.88
NAS' parallel pipeline	3.76 (23.0% reduction)

Table 3.7: Video processing time per phase

In sum, the results indicate that NAS adapts to heterogeneous devices, and a device with higher computational power receives greater benefit.

Temporal variation. We evaluate client's adaptation to temporal variation in computing power in isolation. We emulate the changes in available computing power by varying the clock frequency of GPU (Titan Xp). We change the GPU clock frequency at 10% granularity and report the inference path used and its throughput. Figure 3.15 shows the result compared to a naive version that only uses the full DNN (`NAS-FULL`) and the ideal line that plots the normalized throughput (y-axis) of each inference path at full clock cycle. x-axis shows the number of optional blocks used for inference. The y-intercept represents the computing requirement for the required layers of DNN. We report the raw throughput for all results. It shows NAS adapts to the changing resource with anytime prediction to the extent that the underlying DNN supports and thus delivers real-time performance unlike `NAS-FULL` that does not.

3.5.5 End-to-end Operation

NAS client in operation. We present an end-to-end operation of our NAS client using a network trace from our testset. We use a client with a Titan Xp GPU, running on our local testbed. Figure 3.16 shows the time-line starting from a video request made from our DASH client. At $t = 0.36$ (sec), it downloads the video manifest and test-runs the mock DNNs to select the DNN quality level. The test-run finishes at $t = 2$. The video starts playing at $t = 2.03$. At $t = 17.64$, the first DNN chunk is downloaded, and the minimal DNN initialized at $t = 17.71$. At this time, the DNN processing begins, and video chunks (6-7) in the playback buffer receive quality enhancement. Subsequent video chunks are processed by the DNN as they arrive. As new DNN chunks arrive, the DNNs are incrementally updated. At $t = 46.41$, DNNs are fully downloaded.

DNN processing time. We evaluate the DNN processing latency of NAS client. For DNN processing,

NAS pipelines three processes, each of which respectively handles decoding, super-resolution, and re-encoding. Table 3.7 shows the processing time for each phase for a four-second video chunk. We use GTX 1080 Ti for processing 'Ultra-high' quality DNN using a 30 fps, 240p video as input. We re-encode the DNN's output in H.264 using the fastest option in ffmpeg [29]. This is because the compression factor is not important. The total processing time when each phase is serialized is 4.88 seconds, whereas our pipelined processing takes 3.76 seconds. Considering super-resolution takes 3.69 seconds, the latency overhead of the rest is minimal.

Finally, we measure the client's GPU memory usage for DNN processing. 'Ultra-high', 'High', 'Medium', 'Low' quality DNNs respectively use 3.57 GB, 3.12 GB, 3.05 GB, and 2.99 GB of GPU memory.

3.6 Summary

This chapter provides the first illustration of how neural enhancement can greatly improve video QoE. In particular, we developed a neural-enhanced video delivery system, called NAS, that utilizes client computation to enhance the video quality. Unlike existing video delivery that solely relies on the bandwidth resource, NAS uses client-side computation powered by deep neural networks (DNNs). NAS introduces new system designs to address practical problems in realizing the vision on top of DASH. Our evaluation over real videos on real network traces shows NAS delivers improvement between 21.89–76.04% in user quality of experience (QoE) over the current state of the art. Finally, the cost-benefit analysis shows content distribution networks can actually reduce the cost of video delivery while providing the same or better QoE compared to the current state of the art.

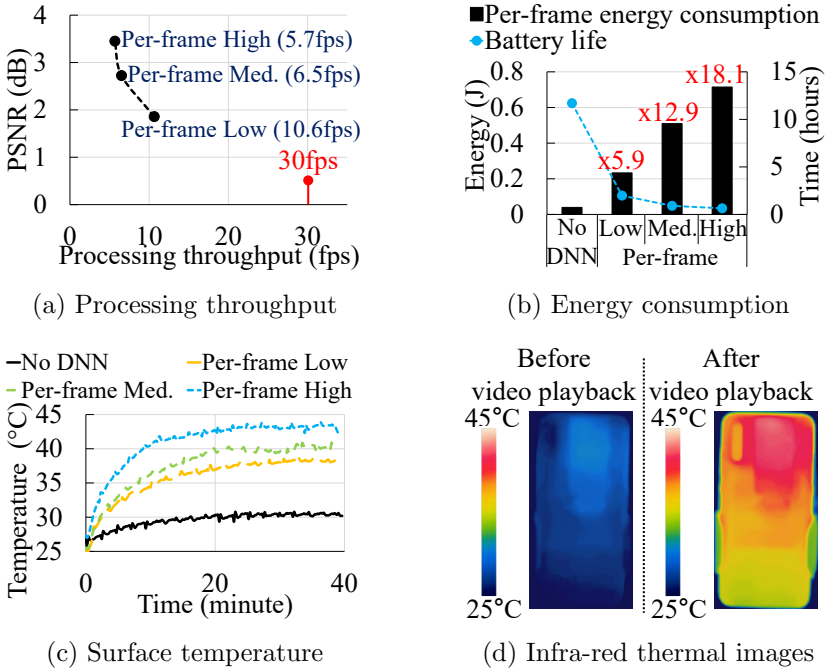
Chapter 4. Neural-enhanced Mobile Streaming

In the previous chapter, we have demonstrated that recent advances in neural-enhanced video streaming create a new opportunity for QoE enhancement using client-side computation independent of the network bandwidth. 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 [212, 211, 137], mobile devices inherently have weak computational capacities coupled with strenuous power constraints. Even the state-of-the-art mobile super-resolution [161], 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 [151] that causes discomfort to users, as we demonstrate in §4.1.

Motivated by these limitations, this chapter presents the illustration of how neural enhancement can be greatly accelerated by leveraging a vast amount of temporal redundancies within a video. In particular, we develop selective inference that applies neural enhancement to a few select frames but reuses the result of super-resolution to benefit the entire video. Selective inference runs in two phases. 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, but the number of possible sets of anchor points are on the order of $2^{|frame|}$, making exhaustive search infeasible. Finally, mobile devices are heterogeneous 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.

To address these challenges, we introduce NEMO [210], which is the first system that enables neural-enhanced video streaming on commodity mobile devices. To effectively transfer the gain of neural super-resolution to successive frames, we leverage fine-grained frame dependencies processed in a video codec [73, 74, 76]. 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



(a) Processing throughput (b) Energy consumption

(c) Surface temperature (d) Infra-red thermal images

Figure 4.1: Motivating measurements on a recent smartphone

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.

This chapter is organized as follows. Section §4.1 and Section §4.2 identifies key challenges in enabling neural-enhanced video streaming and outlines the key design choices to solve these challenges, respectively. Section §4.3 and Section §4.4 describes the detailed design and implementation of NEMO, respectively. Section §4.5 presents evaluation results that illustrates significant throughput improvement by NEMO. Finally, Section §4.6 concludes the chapter.

4.1 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 [151]. To clearly illustrate these challenges, we measure the overhead of running neural super-resolution on a recent smartphone (Xiaomi Mi9 [82]) 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 [85] using Google's Exoplayer [31] 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 [212] (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 4.1(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 [46].[1] Figure 4.1(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 4.1(c) shows the surface temperature of the hottest point over time measured using the FLIR ONE thermographic camera [30]. Figure 4.1(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 [158, 151] 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.

4.2 Key Design Choices

To overcome the limitations of per-frame super-resolution in §4.1, 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.

4.2.1 What to Cache?

Key observations. A DNN consists of multiple layers, and each layer delivers different levels of computation savings and quality loss by caching. Prior studies [208, 143, 107] that focus on object

[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 [48].

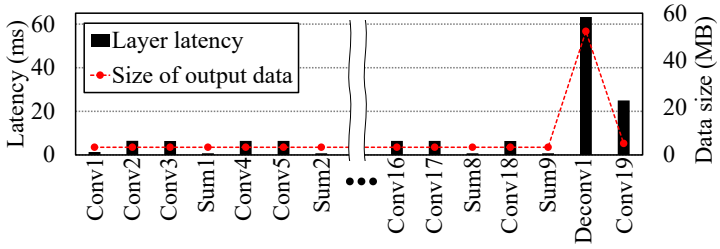


Figure 4.2: Per-layer latency benchmark

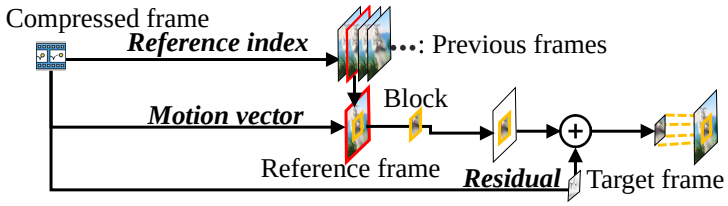


Figure 4.3: Frame dependencies processed in a codec [14]

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 4.2 shows the per-layer latency of a super-resolution DNN (used in NAS [212]) measured using a recent smartphone [82]. 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.

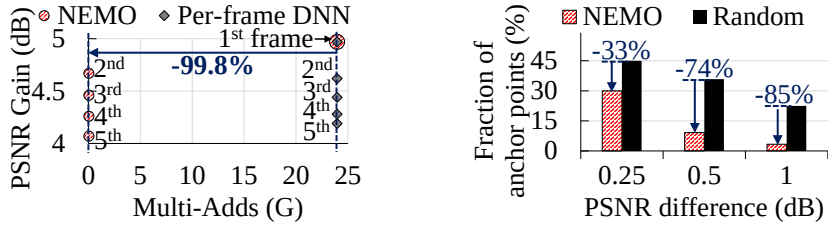
Approach. NEMO applies neural super-resolution only to a subset of frames, referred to as *anchor points*. To make the most out of caching, we cache the final output (i.e., the high-resolution image). The remaining frames reuses the cached result to up-scale their resolutions. To maximize the benefit of reuse, we carefully transfer/reuse the pixels of cached frames using fine-grained frame dependencies (§4.2.2) and select an optimized set of anchor points that delivers the largest overall quality improvement (§4.2.3).

4.2.2 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 [223, 222, 110], running them on resource-constrained mobile devices can largely diminish the computation savings introduced by caching.

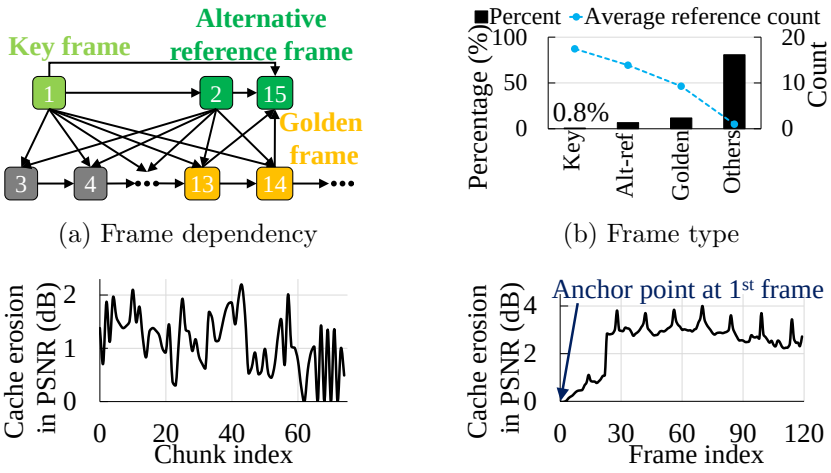
Fortunately, our key observation is that rich information about frame dependencies are embedded in a compressed video and already processed when a video is decoded. Figure 4.3 illustrates how frame dependency information within a codec is used to decode an inter-coded block. First, the codec uses *reference 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



(a) Cache (b) Anchor points

Figure 4.4: Potential benefits of NEMO



(a) Frame dependency (b) Frame type

(c) Cache erosion per chunk (d) Cache erosion per frame

Figure 4.5: Key observations on anchor points

dependencies processed inside a video codec. To make this efficient, we develop an *SR-integrated codec* (§4.3.1) 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.

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 [212] to a 240p video [85]. Figure 4.4(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 [82], we observe that a non-anchor frame is processed very fast (≤15 ms), while an anchor point consumes more than 150 ms.

4.2.3 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 4.5(a) shows the dependency graph for 15 frames within a video [85] encoded with VP9; nodes and edges represent frames and dependencies, respectively. Figure 4.5(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 4.5(c) illustrates the average cache erosion over each (GOP-sized) video chunk within a video [85], 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 4.5(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 $\sim$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 (§4.3.2) 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 4.4(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.

4.3 System Design

Figure 4.6 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-

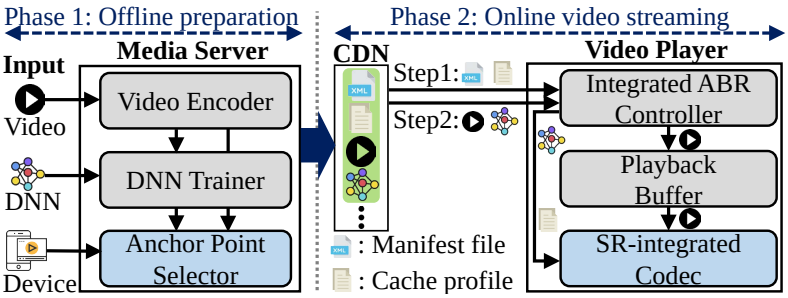


Figure 4.6: NEMO framework overview

bitrate versions and trains super-resolution DNNs using these videos as with content-aware adaptive streaming [212]. 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 (§4.3.2). 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 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 (§4.3.3).

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 [212], 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 (§4.3.1). Finally, super-resoluted frames are rendered by the video player.

4.3.1 SR-Integrated Codec

Goal & Challenge. As compressed videos contain rich information about frame dependencies (§4.2.2), we aim to leverage this codec information to transfer super-resolution outputs over successive frames. However, typical codec implementations [73, 74, 76] 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 [76], AV1 [7]) 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 (§4.2.3) 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 [76]. Figure 4.7 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 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.

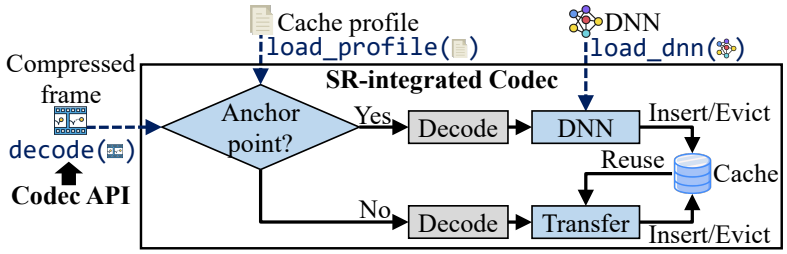


Figure 4.7: SR-integrated codec overview

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:

$$\textbf{decode}(\texttt{compressed frame}) \rightarrow \texttt{super-resoluted frame}$$

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 [31] (see Table 4.1).

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 4.8 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 [6] available on mobile CPUs. However, since bilinear up-sampling results in loss of high-frequency components, cache erosion is inevitably produced. In §4.3.2, 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 §4.3.2).

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.

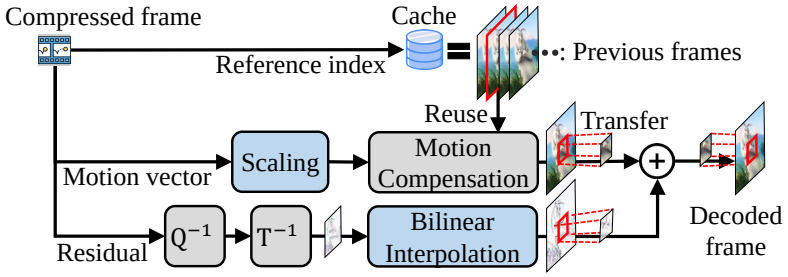


Figure 4.8: Reusing cached high-resolution frames [14] (Q^{-1}, T^{-1}: Inverse-quantization, -transform)

4.3.2 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.

Alternatively, codec-level information can be used to select anchor frames because it affects the effectiveness of anchor frames as discussed in Section §4.2. However, this approach alone cannot guarantee the desired video quality improvement because it does not estimate the resulting video quality. Furthermore, the number of necessary anchor frames varies greatly depending on the dynamics of the video and the difficulty of neural enhancement. As a result, this approach may lead to either a significant decrease in video quality due to a lack of anchor frames, or excessive computational redundancy due to an excessive number of anchor frames.

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 [85], 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,

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 4.2 to select the anchor point that results in the highest video quality.

we can estimate the video quality as:

$$VQ(DNN(\{AP\}) \approx \sum_{i=0}^{|frame|} \max_{f \in \{AP\}} \frac{FQ(i|DNN(f))}{|frame|} \tag{4.1}$$

where $|frame|$ is the total number of frames within a video. Equation 4.1 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 §4.5.

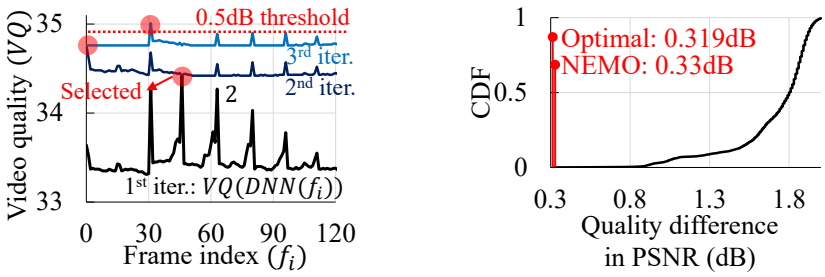
Greedy anchor point selection. Next, we use the quality estimation to iteratively select the most effective anchor point, until the resulting video quality difference from per-frame super-resolution falls within the threshold. NEMO uses Algorithm 1 to process each chunk as follows.

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 4.1 (line 7) in the following way:

$$\begin{aligned} &\max_{AP \in \{F\}} VQ(DNN(AP \cup \{AP'\})) \\ &\approx \max_{AP \in \{F\}} \left(\sum_{i=0}^{|frame|} \max_{f \in \{AP \cup \{AP'\}\}} \frac{FQ(i|DNN(f))}{|frame|} \right) \end{aligned} \tag{4.2}$$

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



(a) Video quality per iteration

(b) Exhaustive search

Figure 4.9: Case study: Anchor point selection

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 §4.2.3. 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 4.9 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 4.2 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 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 4.9(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.

4.3.3 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 [48]. 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

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 4.1: NEMO implementation (Lines of Code)

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 [2]).

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:

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

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 §4.5, 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.

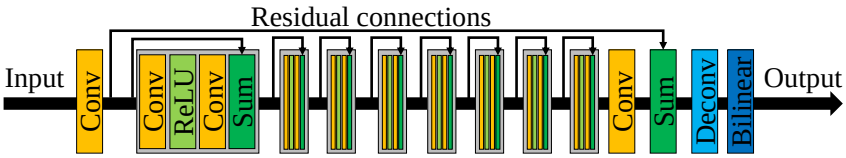


Figure 4.10: Super-resolution DNN architecture

4.4 Implementation

We implement NEMO by extending libvpx [32] and ExoPlayer [31]. Table 4.1 shows the lines of code (LoC) modified.

SR-integrated codec. We extend libvpx (version 1.7.0) [32], 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) [61], 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 [6] available on mobile CPUs.

Video Player. We modify Exoplayer VP9 extension (version 2.9.6) [31] 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 [212] but modify the architecture in two ways to reduce the computing/memory usage. As illustrated in Figure 4.10, 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.

4.5 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?

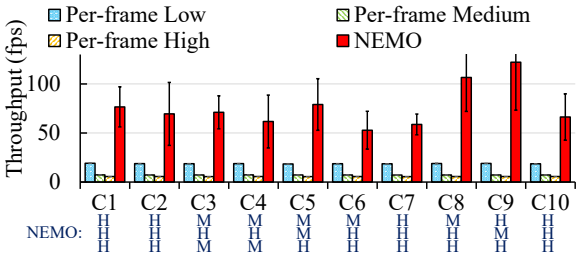


Figure 4.11: Throughput comparison (240p input)

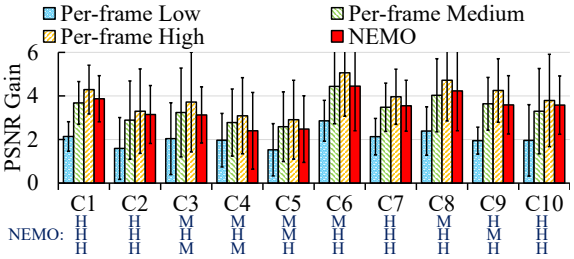


Figure 4.12: Quality gain comparison (240p input)

Mobile Devices. We use three local devices we purchased and four remote devices from Amazon device farm [2]. 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 4.2 presents their specifications. We use Xiaomi Mi9 as the default device unless otherwise noted. We use the Monsoon High Voltage Power Monitor [46] for measuring the energy consumption, and the FLIR ONE thermographic camera [30] for measuring the surface temperature. During all measurements, we set the screen brightness to its maximum.

Video dataset. We use 4K videos from the top ten popular categories [43] 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 [48]. 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 [81]: {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 [48]. Like in NAS [212], our clients apply DNNs to up-scale {240, 360, 480}p videos to 1080p.

4.5.1 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 4.11 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 (§4.3.3), 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. This is because NEMO selects a small fraction of frames as anchor points as shown in Figure 4.13 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

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 4.2: Benchmark on multiple mobiles (SDM: Qualcomm Snapdragon Mobile)

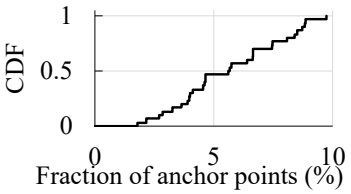


Figure 4.13: Fraction of anchor points

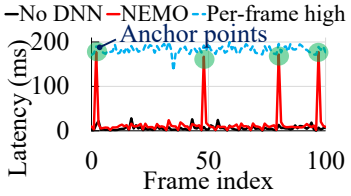


Figure 4.14: Per-frame latency

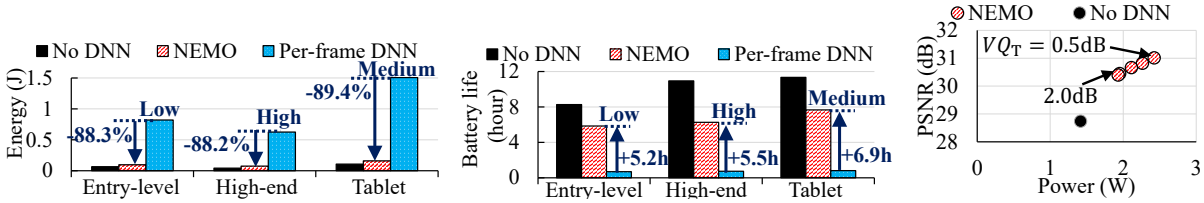
throughput differences. For example, C6 'Haul' gives 52.8 fps, whereas C8 'Favorite' results in 106.6 fps.

Table 4.2 shows the throughput on seven different mobile devices. We use the 'Education' video [83] 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 (§4.3.3). Figure 4.14 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 4.12 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.

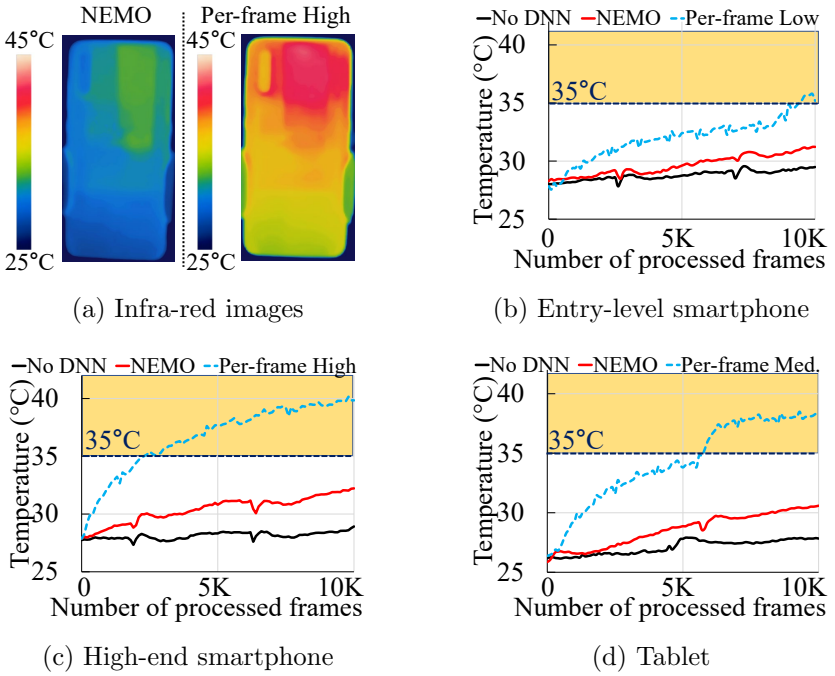
Energy efficiency. Figures 4.15 (a) and (b) show the average per-frame energy consumption and the expected battery life of an entry-level smartphone, a high-end smartphone, and a tablet forthe 'Education' video [83]. NEMO uses 'Low', 'Medium', and 'High' quality DNNs for the entry-level smartphone and 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



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

Figure 4.15: Energy consumption and battery life for 'Education' content (240p input)



(a) Infra-red images (b) Entry-level smartphone

(c) High-end smartphone (d) Tablet

Figure 4.16: Surface temperature comparison for 'Education' content (240p input)

switching cache profiles with different quality thresholds. Figure 4.15(c) shows the trade-off with different thresholds for the 'Education' video [83]. 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 4.16(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 4.16 (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 [158, 151]. 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.

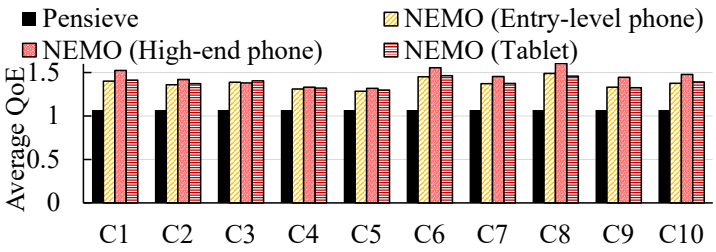


Figure 4.17: QoE improvement on multiple mobiles

4.5.2 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 [170]. 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 [212] that simultaneously streams a DNN and a video. We use the standard QoE metric, used in prior studies [214, 170, 212], 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 [212].

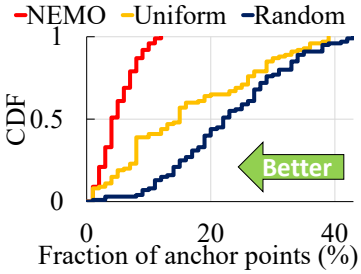
Figure 4.17 shows the average QoE across the ten content categories. As a baseline, we run adaptive streaming using the Pensieve ABR algorithm without applying super-resolution. NEMO consistently outperforms Pensieve for all devices; on average, it delivers 28.7% better QoE for the entry-level, 35.7% 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 untilNEMO'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.

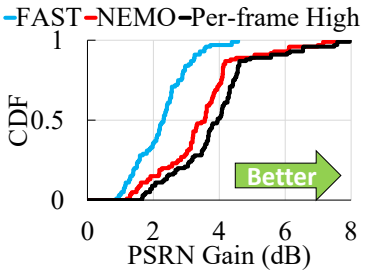
Cost at the server side. We now provide a rough estimate of the server-side cost. Note, the server-side component is not optimized for efficiency, thus the number we provide serves as an upper-bound estimate. First, NEMO trains multiple quality DNNs for each video. The total training time per minute 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 minute of video. Finally, NEMO tests multiple performance options on each mobile. Using the Amazon device farm [2] for this costs $0.4 per minute per device.

4.5.3 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

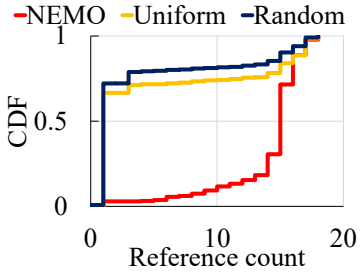


(a) Fraction of anchor points

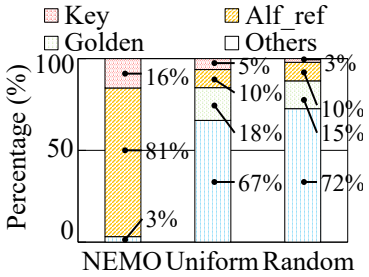


(b) Quality gain

Figure 4.18: NEMO vs. Baselines



(a) Reference count



(b) Frame type

Figure 4.19: Analysis on anchor points

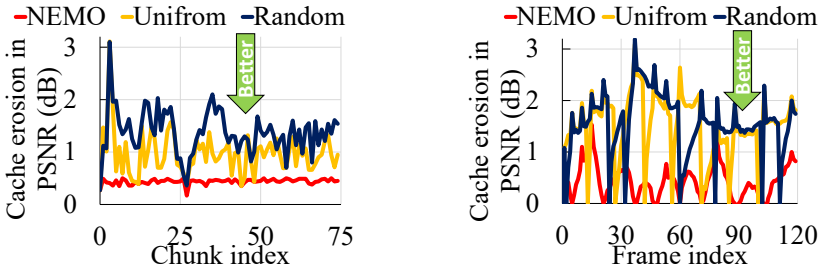
selects randomly-spaced anchor points. Figure 4.18(a) shows the CDF of the average fraction of frames that are anchor points for the threshold of 0.5 dB. Here, we do not limit the number of anchor points, as we do for NEMO in §4.3.3, for a fair comparison. NEMO significantly reduces the number of anchor 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 4.18(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 [218], 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 4.19(a) shows the average reference count of anchor frames and Figure 4.19(b) shows their makeup broken down into VP9 frame types. We use the 'Education' video [83] 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.

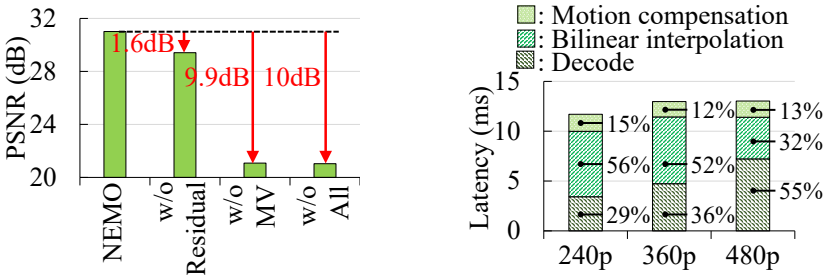
Finally, we demonstrate that NEMO effectively manages cache erosion. Figures 4.20 (a) and (b) show the per-chunk and the per-frame cache erosion of a GOP, respectively. The baselines use the same number of anchor points as in NEMO. NEMO bounds the average cache erosion of each chunk within 0.5 dB, whereas 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



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

Figure 4.20: Analysis on cache erosion



(a) Quality breakdown (b) Latency breakdown

Figure 4.21: Analysis on SR-integrated codec

to the resulting quality by excluding them one at a time in Figure 4.21(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 4.21(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.

4.6 Summary

This chapter illustrates how neural-enhanced video streaming can support mobile clients. In particular, we built NEMO, which 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.

Chapter 5. Neural-enhanced Live Ingest at Scale

In the previous chapters, we have demonstrated that neural enhancement can greatly improve the QoE of on-demand video streaming (at the downstream side) utilizing client computation. In this chapter, we broaden our application to live streaming (at the ingest side) in which we apply neural enhancement to the live ingest video utilizing server computation. While recent advances in neural-enhanced live streaming [52, 153, 201, 168, 109, 65] show great promise in enhancing the ingest video quality, the common limitation is that neural enhancement is too costly to support commercial-scale live streaming. For example, Twitch [88] supports more than 100,000 concurrent live streams [71]. 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 (§5.1). 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 [161], and video encoding is up to 3.3× slower than super-resolution.

Motivated by this limitation, 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 [210] 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 [42, 73, 74] are computationally expensive in compressing high-resolution videos (4K/8K), often becoming a bottleneck in end-to-end neural enhancement (§5.1). 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 (§5.1).

To this end, we present NeuroScaler [213], 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 [210] that requires per-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

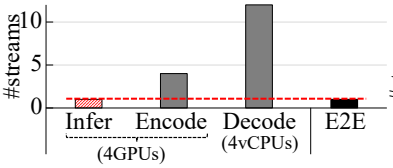


Figure 5.1: Per-frame SR is limited by the inference overhead.

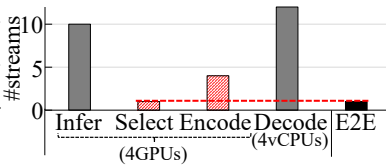


Figure 5.2: Selective SR is limited by the selection/encoding overhead.

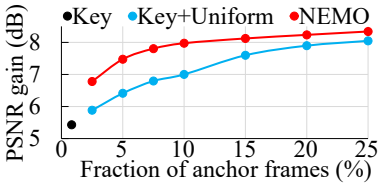


Figure 5.3: Naive anchor selection methods degrade the quality.

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.

This chapter is organized as follows. Section 5.1 describes the computational bottlenecks in end-to-end neural enhancement. Section 5.2, Section 5.3, Section 5.4 illustrate the design of NeuroScaler, anchor frame scheduler, and anchor frame enhancer, respectively. Section 5.5 briefly describes the implementation details of NeuroScaler. Section 5.6 presents evaluation results that illustrates significant throughput improvement by NeuroScaler. Section 5.7 concludes the chapter.

5.1 Challenges

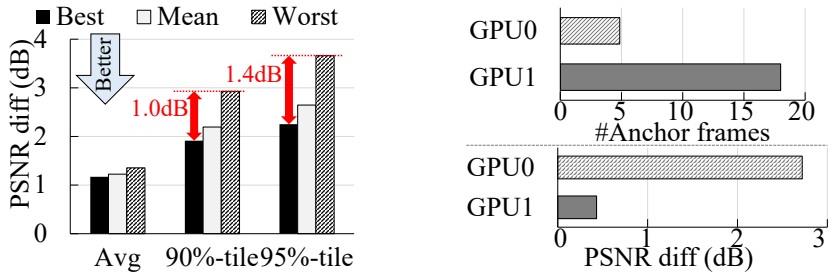
To demonstrate that end-to-end neural enchantment is prohibitively expensive and inefficient, we benchmark its cost and quality gain on a public cloud.

5.1.1 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 [210]. We use a GPU cloud instance [10] that has four inference-optimized NVIDIA T4 GPUs [54]. We run the "high quality" DNN from NAS [212], which up-scales a 720p, 60 fps video [23] 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 5.1 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 [71], this translates to $169,000 per hour.

• Selective SR (NEMO) shows 10× improvement in inference throughput, as shown in Figure 5.2. 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



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

Figure 5.4: Anchor-agnostic resource scheduling results in noticeable quality loss.

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 5.3 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.*

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 [42] and the H.265 hardware encoder [53] on the same instance as above.

Figure 5.2 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 [16, 9, 3, 11]. In this case, running the software encoder [42] is 5× slower than the selective inference. Alternatively, video encoding can be offloaded to (off-chip) video codec FPGAs/ASICs [12, 49]. However, transmitting 4K/8K frames consumes too much bandwidth, even with lossless image compression [59] (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.*

5.1.2 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 [50, 40] 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

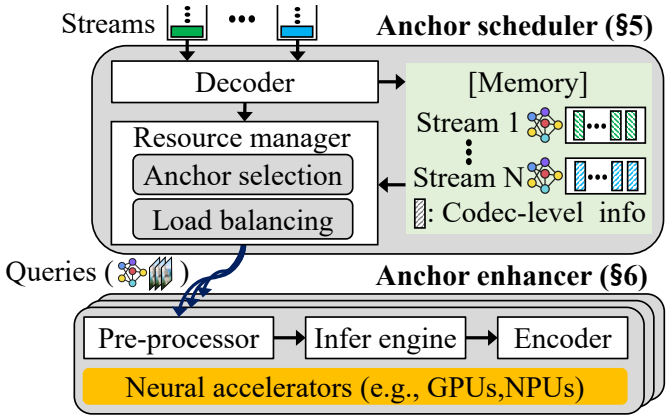


Figure 5.5: NeuroScaler overview

(§5.2) per GPU. We use two servers, each with a single NVIDIA T4 GPU [54]. 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 5.4(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 5.4(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 [50, 40, 26] 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.*

5.2 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. 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 [153] 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 [197, 99, 153].

Overview. NeuroScaler takes an ingest stream and a DNN pair and outputs a high-resolution stream;

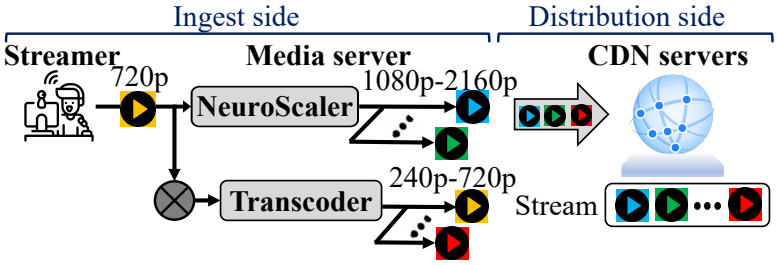


Figure 5.6: NeuroScaler deployment model

the DNN can also be dynamically updated through online learning as in LiveNAS. Figure 5.5 shows the overall workflow of NeuroScaler that consists of the *anchor scheduler* (§5.3) and the *anchor enhancers* (§5.4). 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 5.6 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 [80]. 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.

5.3 Anchor Scheduler

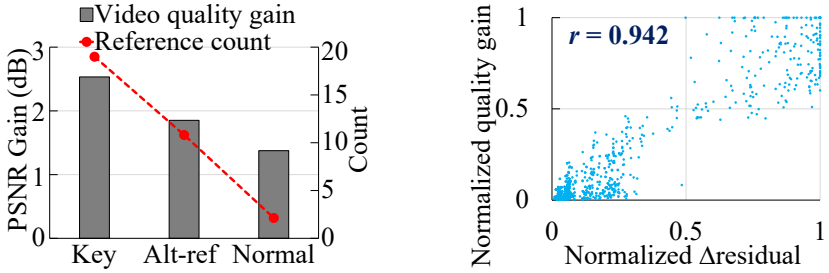
This section describes NeuroScaler's anchor frame selection algorithm (§5.3.1) and resource management modules (§5.3.2) that utilizes the algorithm.

5.3.1 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 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 [76, 7], 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 [84] when using an anchor frame per frame type. Figure 5.7(a)



(a) Frame type (b) Residual

Figure 5.7: Key observations on anchor frames

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 5.1. To demonstrate this, we measure the reduced residual and the video quality gain when using an alternative reference frame as an anchor frame. shows that there is a high correlation between quality gain and the reduced residual; the Pearson correlation coefficient [124] (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 5.8 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, which is explained later in this section. For the frames in G_{key}, we assume they have equal anchor gain as they do 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. The algorithm first calculates accumulated residuals across frames. 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:

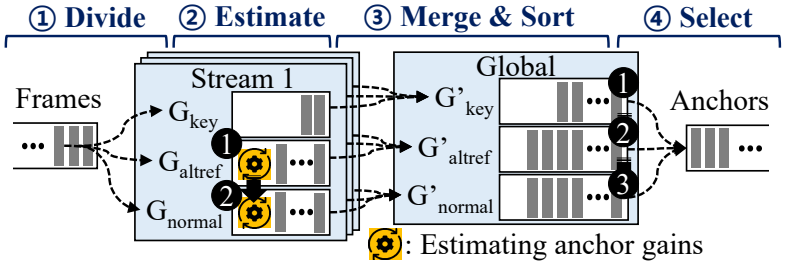


Figure 5.8: Zero-inference anchor selection algorithm

$$\begin{aligned}\Delta\mathrm{Res}(F[i]) &= \sum_{j} \big(\underbrace{\mathrm{Res}(F[j])}_{F[i]\neq\mathrm{Anchor}} - \underbrace{\mathrm{Res'}(F[j])}_{F[i]=\mathrm{Anchor}}\big) \\ &= \sum_{j=i}^{k-1} \big(\mathrm{Res}(F[j]) - (\mathrm{Res}(F[j]) - \mathrm{Res}(F[i]))\big) \\ &= (k-i) \times \mathrm{Res}[i]\end{aligned} \tag{5.1}$$

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.

5.3.2 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.

In particular, the stream-level load balancers in §5.1 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 5.9.

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 (§5.3.1),

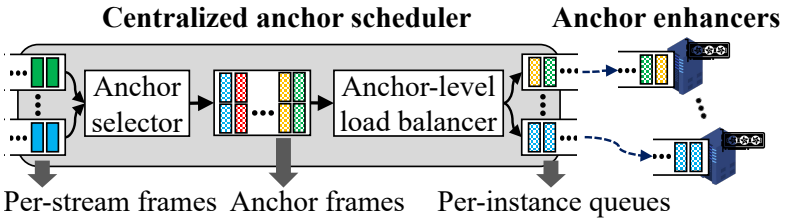


Figure 5.9: Anchor-aware scheduler overview

choosing the largest number of anchor frames that can be processed in real time on a computing cluster as follows:

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

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 but 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.

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 5.14 in §5.6.1. 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 [68].

5.4 Anchor Enhancer

This section describes NeuroScaler's hybrid codec (§5.4.1) and GPU context switching optimizations (§5.4.2).

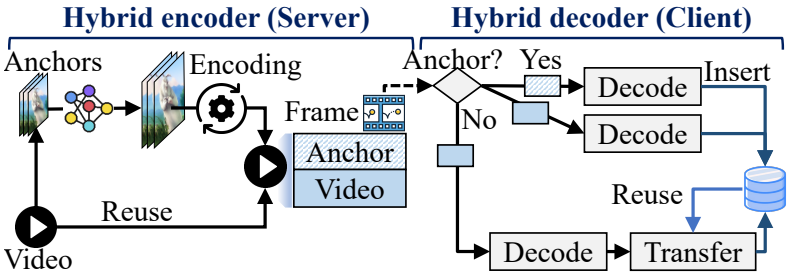


Figure 5.10: Hybrid codec: overall workflow

5.4.1 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 (§5.1). 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 by-passed 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 [210]. 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 5.10 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 (§5.6.2). 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) [104] improves by 6.69% when using the hybrid encoder compared to re-encoding a video using a traditional encoder (§5.6.2).

Hybrid decoding (client-side). The hybrid-encoded stream is decoded at the client, as shown in Figure 5.10. 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 (§4). Decoding a hybrid-encoded stream incurs minimal overhead. Our evaluation (§5.6.2)

shows that the hybrid decoder slightly increases energy consumption by 18% compared to a traditional decoder.

5.4.2 Optimizing GPU Context Switching

We aim to efficiently execute super-resolution DNNs on inference-optimized frameworks (e.g., TensorRT [55], TVM [4]). However, in neural-enhanced live streaming, naively using these frameworks severely degrades the speed due to two types of GPU context switching overhead. First, recent DNN compilers [55, 4, 220, 221] 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 [55], 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 [54]), multiple content-aware DNNs, each requiring several GBs of memory, must be frequently swapped in/out.

Problem (DNN update). Model optimization 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 (§5.2), 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 (§5.6.2).

Problem (DNN loading). Multiple DNNs/frames arrive at each anchor enhancer per scheduling interval (§5.3.2); 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. This scheme makes the overhead negligible regardless of the requested memory size.

5.5 Implementation

NeuroScaler is implemented upon commercial frameworks including libvpx (v1.10) [42], TensorRT (v8.0.3) [69], libjpeg-turbo (v2.1.2) [41], Kakadu H2JTK (v8.2.1) [39], and gRPC (v1.42.0) [33]. 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.

5.6 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?

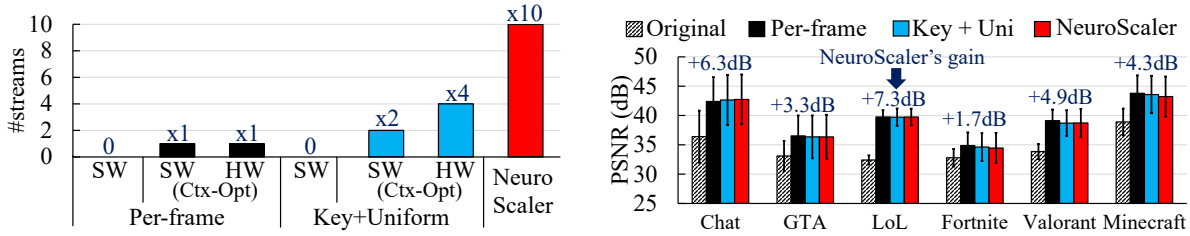
Hardware. Experiments were conducted on AWS EC2 `g4dn` instances [10] that have Intel 2.5 GHz Cascade Lake CPUs and NVIDIA T4 GPUs. We ran neural inference on the GPUs and other processes (e.g., decoding, encoding, anchor frame selection) on the CPUs.
Video. We setup raw videos and encoded them for ingest and distribution as follows:
Setup: We used videos from the top six most-watched content categories on Twitch in 2021 [47]. 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 [20, 23, 25, 22, 21, 24].
Ingest: We used the VP9 codec following the Youtube Live settings [86]. 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). 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 [13, 1].
Distribution: We compressed super-resolved frames as follows. First, for per-frame encoding, the software VP9 codec [42] and the hardware H.265 codec [53] 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 [39] was used.
Super-resolution DNN. We used the "high-quality" DNN from NAS [212], which has 8 residual blocks, 32 channels, and a scale factor of 3. To emulate the benefit of online learning as in LiveNAS [153], 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.

5.6.1 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 [153]) 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.



(a) End-to-end processing throughput

(b) Quality gain of neural enhancement

Figure 5.11: 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)

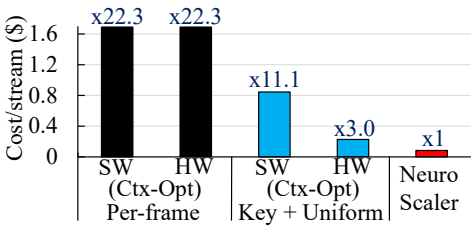


Figure 5.12: End-to-end processing cost based on cost-effective cloud GPU instances

Method	#streams
Key+Uniform SR	10
Ctx-Opt	1
Ctx-Opt + Anchor-Sel	1
Ctx-Opt + Hybrid-Enc	1
Ctx-Opt + Hybrid-Enc + Anchor-Sel	10

Figure 5.13: End-to-end throughput breakdown of NeuroScaler

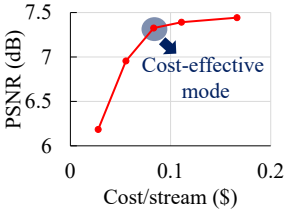


Figure 5.14: Trade-off btw cost and quality in NeuroScaler

Throughput improvement. Figure 5.11(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 (§5.4.2), 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 5.11(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).

Cost saving. To compare the end-to-end processing cost, we select the most cost-effective computing instance for each approach. Figure 5.12 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. NeuroScaler's large cost reduction (up to 22.3x) comes from the fact that NeuroScaler greatly reduces both CPU and GPU usage, which enables

[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.

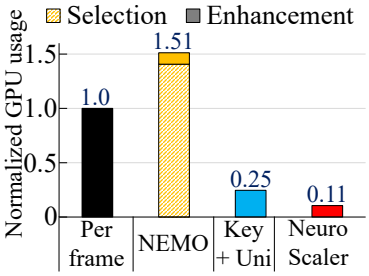


Figure 5.15: SR inference resource usage

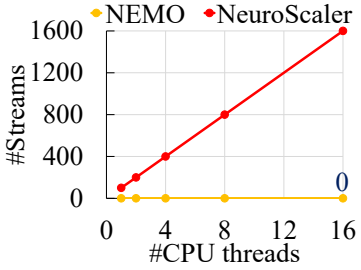


Figure 5.16: Anchor frame selection throughput

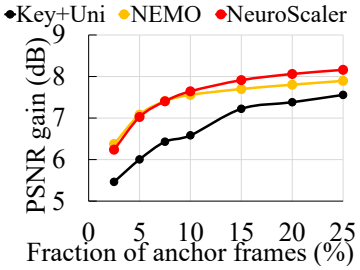


Figure 5.17: Anchor frame efficiency in quality gain

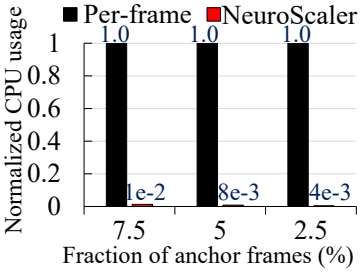


Figure 5.18: Video encoding resource usage

NeuroScaler to run on more economic instance types. The reduced GPU usage is due to selective inference with the zero-inference algorithm (§5.3.1) and the context switching optimization (§5.4.2). The reduction in CPU usage comes from the use of the hybrid codec (§5.4.1). 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 5.11(a).

Contributions by individual components. Figure 5.13 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 (§5.3.1), the hybrid encoding (§5.4.1), and the context switching optimization (§5.4.2). 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 (§5.3.2) 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 [23], as shown in Figure 5.14. 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. 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 [72, 35], the addition latency from neural enhancement has a minimal impact. Second, the latency-critical mode produces delay of

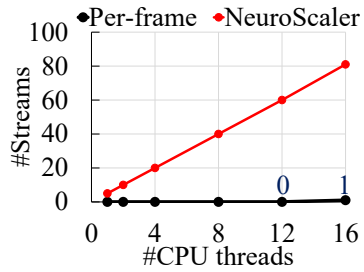


Figure 5.19: Video encoding throughput

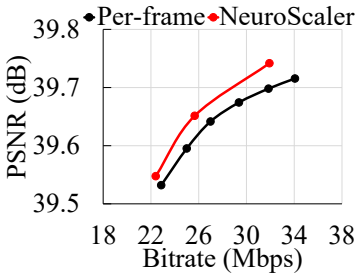


Figure 5.20: Video encoding efficiency (Distribution-side)

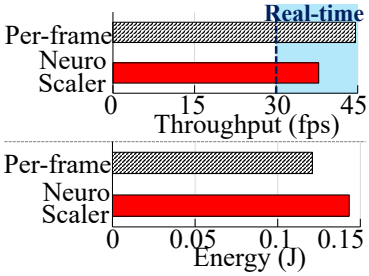


Figure 5.21: Video decoding overhead on a smartphone [82]

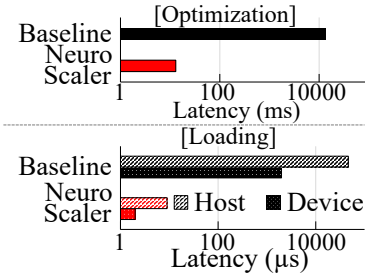


Figure 5.22: GPU context switching overhead

90.8 ms on average (±25.8 ms), which satisfy the delay requirement (200 ms) of video conferencing [68]. 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 [56, 51] which provide lower latency.

5.6.2 Component-wise In-depth Analysis

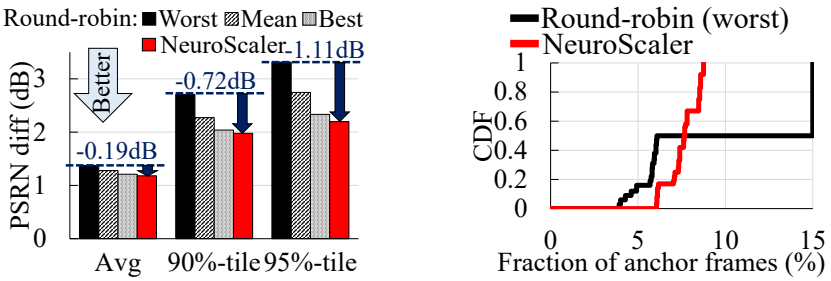
We provide an in-depth performance analysis of individual system components.

NeuroScaler's anchor selector (§5.3.1) 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 5.15 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 5.16 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 5.17 shows the PSNR gain against the fraction of anchor frames for the "League of Legends" video [23]. 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.



(a) Quality difference (b) #anchor frames

Figure 5.23: The benefits of the anchor-level load balancer

NeuroScaler's hybrid codec (§5.4.1) 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 5.18 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 5.19 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.

Compression efficiency: Figure 5.20 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 [104] 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 [82], which has the Qualcomm Snapdragon 855 processor [64]. Figure 5.21 shows the throughput and energy consumption when decoding a 4K, 30 fps video [84] 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 [37]), the hybrid decoder can process a 4K, 60 fps live video with a single thread.

NeuroScaler' GPU context-switching optimization (5.4.2) greatly reduces the overhead of GPU context switching, as shown in Figure 5.22. 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 [60].

NeuroScaler's resource scheduler (§5.3.2) maximizes the quality gain of selective inference. To demonstrate this, we measure the quality difference from per-frame super-resolution for each video chunk.

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 5.24: NeuroScaler's anchor scheduler scalability

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 [10], 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 5.23(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 5.23(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 (§5.1). 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.

5.6.3 Scalability

Scheduler. NeuroScaler's anchor scheduler (§5.2), which consists of the decoder and the resource manager, can operate at scale. Figure 5.24 shows the cost and the latency of each module using the `c6i.32xlarge` instance [8] 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 5.8) are the main computational bottlenecks, but both of them can be parallelized on multiple CPUs.

Case study (Twitch [91]). We estimate the operating cost for running NeuroScaler on a Twitch-scale service, which has 100,000 concurrent live streams. Figure 5.25 quantifies the cost using AWS EC2 instances [8, 10]. NeuroScaler's neural enhancement costs $7,898 per hour, which is 21.3× less expensive than the per-frame counterpart as used in LiveNAS [153]. The anchor scheduler and enhancer accounts for 4.3% and 95.7% of the overall cost, respectively.

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 5.25: NeuroScaler's cost for a Twitch-scale service

5.7 Summary

In this chapter, we looked at the scalability issues in neural-enhanced live streaming at the ingest side. 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.

Chapter 6. Future Work

The potential for neural-enhanced video streaming is vast, but it is still in the early stages of development. This dissertation represents a first step in the direction of exploring this promising area. The emergence of relevant ecosystems in the industry suggests that there will be significant opportunities for neural-enhanced video streaming. For instance, Microsoft has recently announced the native support of video super-resolution in the Edge web browser to enhance the streaming experience [44], and NVIDIA has developed an SDK to facilitate the integration of super-resolution with existing internet applications [58]. The advancements in both academia and industry lead us to expect that neural-enhanced video streaming will play a pivotal role in the emerging media of the next generation (e.g., AR, VR), which are too data-intensive for the current Internet infrastructure to support.

This section proposes two research directions aimed at enhancing the practicality and wider applicability of neural-enhanced video streaming. The first direction involves systemic and algorithmic optimizations that could significantly reduce the operating cost of neural-enhanced video streaming (§6.1). The second direction pertains to the broader applications of neural-enhanced video streaming (§6.2).

6.1 Towards Cost-effective Neural-enhanced Streaming

DNN training. Our work has utilized a content-aware deep neural network (DNN) per video to produce reliable high-quality outcomes. Although this approach enhances the net benefit for popular videos, as demonstrated in NAS [212], the cost of training content-aware DNNs remains considerable. Hence, reducing the training cost is critical to making neural-enhanced video streaming more practical and facilitating its deployment across diverse applications. We have identified two promising research directions to achieve this goal. Firstly, there is significant temporal redundancy across videos, as the same characters and scenes are often repeated in videos produced by the same streamer or videos with similar content (e.g., computer games, sports games). Thus, DNNs retrained on similar content can be reused with minimal fine-tuning. As opposed to creating/serving a DNN per video, extending the granularity from video-level to cluster-level has enormous potential to reduce the overall training cost. This research direction involves creating a new cluster, mapping a new video to an existing cluster, and allocating computing resources across clusters. Secondly, the benefit of neural enhancement significantly varies across videos. We have observed that some videos with a large number of dynamic scenes and noises derive less benefit from neural enhancement. This implies that applying neural enhancement to every video is redundant in terms of cost-effectiveness. To address this issue, a classification method needs to be developed to quickly determine whether a target video is worth applying DNNs. This research direction will allow computing resources to be used more efficiently by focusing on videos that are favorable for neural enhancement.

DNN architecture. Our research has mainly focused on super-resolution as a neural enhancement method, as it offers a high-quality improvement per computation and can be seamlessly integrated with adaptive streaming, which is the de facto standard in current internet video delivery infrastructure. However, we have observed significant advancements in DNN architectural designs for neural enhancement that can substantially enhance the benefits of neural-enhanced video streaming. Specifically, we believe

that video re-scaling and sandwich video compression are two promising components that will propel neural enhancement to the next level. Re-scaling [207, 181, 111] involves jointly learning down-scaling and up-scaling (i.e., super-resolution), which demonstrates superior performance compared to super-resolution on its own. On the other hand, sandwich video compression [128, 129, 145] can be viewed as a generalised version of re-scaling, which aims to develop optimal pre-processing and post-processing for traditional video codecs. These algorithms aim to find the best way to transform data to be favourable for both traditional codecs and DNN-based quality enhancement.

Video compression. Our work has used traditional compression algorithms such as H.26x [73, 74] and VPx [75] to build our solutions based on existing video streaming infrastructure. However, we believe there is ample opportunity to enhance compression efficiency by jointly optimizing neural enhancement and video compression. Existing compression algorithms remove high-frequency redundancy that is nearly imperceptible to the human eye, but do not take into account the potential reconstruction facilitated by DNNs. Given that the effectiveness of DNN-based quality enhancement varies across both space and time, a video compression algorithm can leverage this heterogeneity to adapt its bitrates to maximize the overall neural-enhanced quality within a given size constraint. There are various design considerations that need to be taken into account, such as re-designing an ABR bitrate ladder to optimize QoE for many users, optimizing bitrate allocation within and across frames, and developing a new color space (e.g., YUV), frequency domain transformation (e.g., Discrete Cosine Transform), and quantization method (i.e., quantization table) that are favorable for neural enhancement. Since there are numerous factors that affect system performance, we advocate for a holistic view and vertically-integrated optimizations.

6.2 Applying Neural-enhanced Streaming to Broader Services

ML applications. Recently, a significant amount of video data is consumed by machines to provide valuable statistics to humans. For instance, Microsoft FarmBeats [28] aims to maximize the profit of farmers by analyzing videos captured from farms, while Microsoft Rocket [45] automates human labor by analyzing videos from factory floors, traffic intersections, police vehicles, and retail shops. However, video capturing devices in these scenarios often have limited power consumption and/or restricted network bandwidth [136], making low-resolution streaming unavoidable, which severely affects the quality of the analytic pipeline. We believe that neural enhancement can be of great benefit to a wide range of ML applications, similar to how our work has demonstrated for traditional video streaming that targets human viewers. As such, there is a need to redesign existing neural-enhanced video techniques to cater to these applications, as the impact of quality (or quality of experience) differs significantly between humans consuming visual content and machines running analytic services. This involves customizing DNN-based quality enhancement for a specific ML application, designing the corresponding adaptive bitrate control, and scheduling network and computing resources, considering the capabilities of end devices, to maximize end-to-end performance.

Video storage. Our research has focused primarily on video streaming, but we recognize that neural enhancement has the potential to revolutionize video storage by enabling higher compression efficiency. We envision a neural-enhanced video storage system that seamlessly incorporates DNN-based quality enhancement to reduce storage requirements while maintaining the same level of quality. The main challenge in achieving this is to accurately estimate the low-quality bitrate that will result in identical

quality after neural enhancement. A notable example of a similar approach is Lepton [134], which transparently compresses JPEG-encoded images stored in Dropbox by optimizing lossless compression algorithms. A neural-enhanced storage system would provide a similar capability to Lepton, but with significantly greater compression benefits enabled by DNNs.

Emerging media. Our research has been primarily within the scope of 2D video streaming, however, the emergence of new types of videos such as VR, Depth, and 3D videos have become increasingly popular. These types of videos require a large amount of data and processing power, which the current internet infrastructure struggles to support. Thus, we believe that neural-enhanced video delivery will play a crucial role in democratizing emerging media. To apply neural enhancement to emerging media, we need to consider two critical differences between traditional and emerging media. Firstly, 3D video has various representations such as point cloud and reality mesh [130], each with a trade-off between data footage and processing overhead. Therefore, DNN architecture should be re-designed accordingly, and joint optimization of DNNs and video representation should be considered with computing and network resources. Secondly, 3D video has a different form of QoE metric [215], as only a part of the video is shown to humans, and sensitivity to different depths varies greatly. Hence, there is a great potential for optimizing DNN computations to focus on critical parts that directly impact user QoE.

Chapter 7. Conclusion

This dissertation introduces a new paradigm in video streaming called *neural-enhanced video streaming*, which effectively reduces the tight dependency between network resources and video quality. We started by identifying the fundamental challenges in enabling neural-enhanced video streaming, including providing reliable quality enhancement and reducing excessive computing overhead. We then addressed these challenges with domain-specific insights into deep neural networks and video streaming, such as the content-awareness of DNNs and the temporal redundancy within video. Finally, we built three full-fledged systems incorporating these insights to demonstrate how neural-enhanced video streaming can significantly improve user quality of experience (QoE) for video streaming. This chapter concludes the dissertation by summarizing the contributions of each system.

- NAS is the first end-to-end video streaming system that integrates neural enhancement on top of an existing adaptive streaming framework. It addresses several challenges, such as reliable quality enhancement, client heterogeneity, interaction with bitrate adaptation, and DNN transfer. NAS can improve the average QoE by 43.08% with the same bandwidth budget, or save 17.13% of bandwidth while providing the same user QoE.
- NEMO is the first system that enables real-time neural enhancement on commodity mobile devices. NEMO minimizes the overhead of neural enhancement by selectively applying it to a minimal number of frames based on codec-level information of fine-grained frame dependencies. NEMO can improve the overall processing throughput by 15.2×, reduce energy consumption by 88.7%, and maintain device temperatures at acceptable levels, while ensuring high video quality.
- NeuroScaler is the first framework that delivers efficient and scalable neural enhancement for live video ingestion. To accelerate end-to-end neural enhancement, NeuroScaler proposes novel algorithms and system designs that significantly reduce the overhead of selective super-resolution, video encoding, and GPU context switching. Using a public cloud, NeuroScaler respectively improves processing throughput by 2.5-5× and reduces overall cost by 3.0-11.1×.

Acknowledgments in Korean

지난 6년간 KAIST에서의 박사 과정은 제 인생에서 가장 큰 도전이었습니다. 어려웠던 순간이 너무나도 많아 정말 힘들었지만, 주변에서 많은 도움을 얻은 덕분에 성공적으로 박사학위를 취득할 수 있었습니다. 제게 학위 관련하여 도움을 주신, 앞으로 나아갈 용기와 힘을 주신 모든 분께 감사의 마음을 전합니다.

우선 세계적인 연구자로서 발돋움할 수 있게 성심성의껏 도와주신 한동수 교수님께 감사드립니다. 교수님께 연구하는 방법, 연구자로서 자세 및 마음가짐, 어려운 문제를 헤쳐나가는 방법, 힘든 시간을 이겨내는 방법, 남들과 글과 발표를 통해 효과적으로 소통하는 방법 등 앞으로 인생을 살아가는 데 중요한 가르침을 너무나도 많이 받았습니다. 교수님을 만나지 못했더라면 이렇게 아무것도 모르던 학부생에서 짧은 시간 안에 국제적인 수준의 연구자로서 나아갈 수는 없었을 것이라, 교수님의 지도를 받을 수 있었던 것은 제게 너무나도 큰 축복이고 행운이었다고 생각합니다. 앞으로 더 크게 성장하고 선한 영향력을 행사하는 사람이 되어 교수님께 받은 가르침을 더 많은 사람에게 전달하여 자랑스러운 연구실 동문이 되도록 노력하겠습니다.

제 박사 학위 심사를 위해 바쁜 시간을 쪼개어 참석해주고 유용한 피드백을 주신 신진우 교수님, 박경수 교수님, 이성주 교수님, 황의종 교수님, Junchen Jiang 교수님께도 감사의 말씀 드립니다. 신진우 교수님께 학부생 때 인턴십, 대학원 진로 관련하여 도움을 받았기에 이 자리에 올 수 있었고, 제 첫 번째 논문인 NAS를 작성할 때 AI 측면에서 다방면으로 조언 주셔서 컴퓨터 시스템 최고의 학회인 OSDI에 붙을 수 있었습니다. 박경수 교수님께 대학원 네트워크 수업을 들으면서 다방면으로 지식을 쌓아나간 게 연구 능력을 키워나가는데 많은 도움이 됐습니다. 이성주 교수님께 대학원, 연구, 진로 관련해서 다방면으로 유용한 조언을 많이 얻어서 학위 과정에 큰 도움이 되었습니다. 마지막으로 황의종 교수님께서 다른 연구실 학생임에도 OSDI 학회 발표 연습 때 성심성의껏 조언 주셔서 덕분에 잘 마무리 지을 수 있었습니다.

대학원 과정이 너무나도 힘들어 중간에 지치고 주저앉고 싶을 때가 많았지만, 항상 멀리서 묵묵히 응원해주시고 격려해주시고 걱정을 덜어주시고 힘과 용기를 주시는 부모님이 계셔서 한발 한발 계속 나아갈 수 있었습니다. 그리고 삶에서 항상 한발, 두 발 먼저 앞서 나아가면서 그 과정에서 겪은 소중한 정보를 동생에게 공유해주고 언제나 진심 어린 조언을 주는 형에게도 정말 감사합니다. 너무나도 든든하고 화목한 가족 아래에서 성장할 수 있었던 게 제 삶에서 가장 큰 축복이라 생각합니다. 졸업 후에는 부모님께 더 효도하고, 형에게 더 자주 연락하고, 자랑스러운 행복하게 사는 그리고 덜 걱정을 끼치는 아들이 될 수 있도록 더욱 노력하겠습니다.

연구실 생활하면서 항상 도움을 많이 받아왔었는데 INA 연구실 선배, 후배분들에게 감사를 표합니다. 병권이형, 성민이형, 정민이형, 주형이형, 인호형 덕분에 연구실에 잘 적응하고 연구를 어떻게 하는지 옆에서 지켜보면서 잘 배울 수 있었습니다. 논문을 쓰면서 힘든 시간을 동고동락했었던 영목이, 재홍이, 휘준이, 진우, 준철이, 찬주 학생이 있었기에 성공적으로 논문을 낼 수 있었고 이 자리까지 올 수 있었습니다. 앞으로 우리 연구실을 이끌어나갈 윤헌이, 예찬이, 의택이, 재형이, 진영이에게도 앞으로 대학원 생활을 성공적으로 해나갈 수 있게 항상 응원하겠습니다. 이외에 이름을 다 적을 수 없지만, 학사, 박사 과정 동안 KAIST에서 정말 소중한 추억을 함께하고 즐거운 학교생활을 할 수 있게 도와준 새터 1반 친구들, 울산과학고 친구들에게도 고마움을 담아 감사의 글을 마칩니다.

Bibliography

[1] 720p is the preferred broadcasting resolution on twitch. https://parsec.app/blog/720p-is-the-preferred-broadcasting-resolution-on-twitch-e5385a3ef08f.

[2] Amazon Device Farm Official Website. https://aws.amazon.com/device-farm/?nc1=h_ls.

[3] Amazon ec2 inf1 instances website. https://cloud.google.com/tpu.

[4] Apache TVM Official Website. https://tvm.apache.org/.

[5] Apple HTTP live streaming official homepage. https://developer.apple.com/streaming/.

[6] ARM Neon Overview. https://developer.arm.com/architectures/instruction-sets/simd-isas/neon.

[7] AV1 Specification. https://aomedia.org/av1-features/get-started/.

[8] AWS EC2 C6i Instance Official Website. https://aws.amazon.com/ko/ec2/instance-types/c6i/.

[9] AWS EC2 DL1 Instance Official Website. https://aws.amazon.com/ec2/instance-types/.

[10] AWS EC2 G4 Instance Official Website. https://aws.amazon.com/ec2/instance-types/g4/.

[11] AWS EC2 P4Instance Official Website. https://aws.amazon.com/ec2/instance-types/p4/.

[12] Aws f1 instance website. https://aws.amazon.com/ec2/instance-types/f1/.

[13] The best streaming bitrate and resolutions for twitch. https://bit.ly/32rX2Pe.

[14] Big Buck Bunny. https://peach.blender.org/.

[15] Binge boom: Young u.s. viewers gulp down average of six tv episodes per sitting. https://variety.com/2017/digital/news/binge-viewing-tv-survey-millennials-1202013560/.

[16] Cloud tpu website. https://aws.amazon.com/ec2/instance-types/inf1/.

[17] Dash industry forum. https://dashif.org/.

[18] Dash recommend segment length. https://bitmovin.com/mpeg-dash-hls-segment-length/.

[19] dash.js Github repository. https://github.com/Dash-Industry-Forum/dash.js.

[20] Dataset: Chat youtube video. https://www.youtube.com/watch?v=TJoAy944MoM.

[21] Dataset: Fortnite youtube video. https://www.youtube.com/watch?v=LW4asVQsew0.

[22] Dataset: Gta5 youtube video. https://www.youtube.com/watch?v=jYglXGpaBSg.

[23] Dataset: League of legend youtube video. https://www.youtube.com/watch?v=Ku_q0O_kgGE.

[24] Dataset: Minecraft youtube video. https://www.youtube.com/watch?v=5gff3AGv7o8.

[25] Dataset: Valorant youtube video. https://www.youtube.com/watch?v=dqhVl11DypA.

[26] Deploying nvidia triton at scale with mig and kubernetes. https://bit.ly/31X8zVQ.

[27] Draft VP9 Bitstream and Decoding Process Specification. https://www.webmproject.org/vp9/.

[28] Farmbeats: Ai, edge & iot for agriculture. https://www.microsoft.com/en-us/research/project/farmbeats-iot-agriculture/.

[29] FFmpeg - H.264 Video Encoding Guide. https://trac.ffmpeg.org/wiki/Encode/H.264.

[30] FLIR Official Website. https://www.flir.com/.

[31] Google's Exoplayer Official Website. https://developer.android.com/guide/topics/media/exoplayer.

[32] Google's libvpx Official Github Repository. https://github.com/webmproject/libvpx/.

[33] gRPC Github Repository. https://github.com/grpc/grpc.

[34] H.264 : Advanced video coding for generic audiovisual services .

[35] How long is twitch stream delay. https://onetwostream.com/blog/twitch-delay/.

[36] How to ensure a full live-streaming experience. https://www.techradar.com/news/how-to-ensure-a-full-live-streaming-experience.

[37] 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.

[38] The iPhone X's new neural engine exemplifies Apple's approach to AI. https://www.theverge.com/2017/9/13/16300464/apple-iphone-x-ai-neural-engine.

[39] Kakadu official website. https://kakadusoftware.com/.

[40] Kubernetes load balancer document. https://kubernetes.io/docs/concepts/services-networking/_print/.

[41] libjpeg-turbo github repository. https://github.com/libjpeg-turbo/libjpeg-turbo.

[42] libvpx github repository. https://github.com/webmproject/libvpx.

[43] 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.

[44] Microsoft Edge is getting a video upscaler to make blurry old videos look better. https://www.theverge.com/2023/3/6/23627080/microsoft-edge-video-upscaling-ai-super-resolution-vsr.

[45] Microsoft rocket for live video analytics. https://www.microsoft.com/en-us/research/project/live-video-analytics/.

[46] Monsoon Official Website. https://www.msoon.com/.

[47] Most watched games on twitch in 2020. https://sullygnome.com/games/2020/watched.

[48] NEMO's official Github Repository. https://github.com/kaist-ina/nemo.

[49] Ngcodec uses amazon ec2 f1 instances with custom fpgas running 4k video compression. www.prlog.org/12604725.

[50] Nginx load balancer document. https://docs.nginx.com/nginx/admin-guide/load-balancer/http-load-balancer/.

[51] NVIDIA A100 Official Website. https://www.nvidia.com/en-us/data-center/a100/.

[52] NVIDIA Maxine Official Website. https://developer.nvidia.com/maxine.

[53] Nvidia nvenc documentation. https://docs.nvidia.com/video-technologies/video-codec-sdk/ffmpeg-with-nvidia-gpu/.

[54] NVIDIA T4 Official Website. https://www.nvidia.com/en-gb/data-center/tesla-t4/.

[55] NVIDIA TensorRT Official Website. https://developer.nvidia.com/tensorrt.

[56] NVIDIA V100 Official Website. https://www.nvidia.com/en-gb/data-center/tesla-v100/.

[57] Pensieve official Github repository. https://github.com/hongzimao/pensieve.

[58] Pixel Perfect: RTX Video Super Resolution Now Available for GeForce RTX 40 and 30 Series GPUs . https://blogs.nvidia.com/blog/2023/02/28/rtx-video-super-resolution/.

[59] Png standard website. https://www.iso.org/standard/29581.html.

[60] Pytorch official website. https://pytorch.org/.

[61] Qualcomm Snapdragon Neural Processing Engine Official Website. https://developer.qualcomm.com/docs/snpe/index.html.

[62] Raw Data - Measuring Broadband America 2016. https://www.fcc.gov/reports-research/reports/measuring-broadband-america/raw-data-measuring-broadband-america-2016.

[63] Scalable video coding (svc) extension for webrtc. https://www.w3.org/TR/webrtc-svc/.

[64] Snapdragon 855+/860 Mobile Platform. https://www.qualcomm.com/products/snapdragon-855-plus-and-860-mobile-platform.

[65] Softbank solves key mobile edge computing challenges using nvidia maxine. https://bit.ly/3dJNq47.

[66] The state of online video 2017. https://www.limelight.com/resources/white-paper/state-of-online-video-2017/#weeklyconsumption.

[67] Streaming Ad Revenue to Double by 2026. https://yhoo.it/329mAjP.

[68] 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/.

[69] Tensorrt official website. https://developer.nvidia.com/tensorrt.

[70] Twitch Revenue and Usage Statistics (2022). https://www.businessofapps.com/data/twitch-statistics/.

[71] Twitch Statistics. https://backlinko.com/twitch-users.

[72] Twitch stream delay: Everything you should know. https://bit.ly/3HGbPpI.

[73] Video codec - H.264 standardization. https://www.itu.int/rec/T-REC-H.264/.

[74] Video codec - H.265 standardization. http://x265.org/.

[75] Video codec - VP9 official website. https://www.webmproject.org/vp9/.

[76] Webm Official Website. https://www.webmproject.org/.

[77] Webrtc 1.0: Real-time communication between browsers. https://www.w3.org/TR/webrtc/.

[78] WebRTC Official Website. https://webrtc.org/.

[79] WHAT AUDIENCES EXPECT FROM LIVE VIDEO. https://lp.livestream.com/rs/582-GOU-684/images/NYmag.pdf.

[80] Wowza transcoding documents. https://www.wowza.com/docs/wowza-transcoder.

[81] Wowza's DASH bitrate recommendation. https://www.wowza.com/docs/how-to-encode-source-video-for-wowza-streaming-cloud.

[82] Xiaomi Mi9 Specifications. https://www.gsmarena.com/xiaomi_mi_9-9507.php.

[83] YouTube dataset (Education 1). https://www.youtube.com/watch?v=0eaf6bUMd4U.

[84] YouTube dataset (Product review1). https://www.youtube.com/watch?v=ZNC8XT93rds.

[85] YouTube dataset (Unboxing). https://www.youtube.com/watch?v=l0DoQYGZt8M.

[86] Youtube live encoding guidelines. https://support.google.com/youtube/answer/2853702?hl=en.

[87] Youtube recommended upload encoding settings. https://support.google.com/youtube/answer/1722171?hl=en.

[88] Youtube website. https://www.twitch.tv/.

[89] Everything Wrong With Frozen In 10 Minutes Or Less. https://www.youtube.com/watch?v=HvwMtWkfkJ8, June 2014.

[90] Minecraft Xbox - School Day [244]. https://www.youtube.com/watch?v=5N6E2cF-CGw, November 2014.

[91] New twitch rankings: Top games by esports and total viewing hours. https://newzoo.com/insights/articles/new-twitch-rankings-top-games-esports-total-viewing-hours/, July 2016.

[92] Shooting survivor confronts NRA spokesperson Dana Loesch. https://www.youtube.com/watch?v=4AtOU0dDXv8, February 2018.

[93] V. K. Adhikari, Y. Guo, F. Hao, V. Hilt, Z. L. Zhang, M. Varvello, and M. Steiner. Measurement study of netflix, hulu, and a tale of three cdns. *IEEE/ACM Transactions on Networking (ToN)*, 23(6):1984–1997, Dec 2015.

[94] Eirikur Agustsson and Radu Timofte. Ntire 2017 challenge on single image super-resolution: Dataset and study. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, July 2017.

[95] Namhyuk Ahn, Byungkon Kang, and Kyung-Ah Sohn. Fast, accurate, and lightweight super-resolution with cascading residual network. *arXiv preprint arXiv:1803.08664*, 2018.

[96] Saamer Akhshabi, Ali C Begen, and Constantine Dovrolis. An experimental evaluation of rate-adaptation algorithms in adaptive streaming over http. In *Proceedings of the ACM Multimedia Systems Conference (MMSys)*, pages 157–168. ACM, 2011.

[97] Zahaib Akhtar, Y. S. Nam, Sanjay Rao, and Bruno Ribeiro. Oboe : Auto-tuning video abr algorithms to network conditions. In *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM)*, 2018.

[98] Duin Baek, Mallesham Dasari, Samir R. Das, and Jihoon Ryoo. 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, page 336–343, New York, NY, USA, 2021. Association for Computing Machinery.

[99] Duin Baek, Mallesham Dasari, Samir R Das, and Jihoon Ryoo. dcsr: Practical video quality enhancement using data-centric super resolution. 2021.

[100] Athula Balachandran, Vyas Sekar, Aditya Akella, Srinivasan Seshan, Ion Stoica, and Hui Zhang. Developing a predictive model of quality of experience for internet video. *ACM SIGCOMM Computer Communication Review*, 43(4):339–350, 2013.

[101] Athula Balachandran, Vyas Sekar, Aditya Akella, Srinivasan Seshan, Ion Stoica, and Hui Zhang. Developing a predictive model of quality of experience for internet video. In *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM)*, 2013.

[102] Johannes Ballé, David Minnen, Saurabh Singh, Sung Jin Hwang, and Nick Johnston. Variational image compression with a scale hyperprior. In *International Conference on Learning Representations*, 2018.

[103] Yoshua Bengio. Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, 2(1):1–127, 2009.

[104] Gisle Bjontegaard. Calculation of average psnr differences between rd-curves. *VCEG-M33*, 2001.

[105] Tolga Bolukbasi, Joseph Wang, Ofer Dekel, and Venkatesh Saligrama. Adaptive neural networks for efficient inference. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 527–536, 2017.

[106] Jose Caballero, Christian Ledig, Andrew Aitken, Alejandro Acosta, Johannes Totz, Zehan Wang, and Wenzhe Shi. Real-time video super-resolution with spatio-temporal networks and motion compensation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4778–4787, 2017.

[107] Lukas Cavigelli, Philippe Degen, and Luca Benini. Cbinfer: Change-based inference for convolutional neural networks on video data. In *Proceedings of the 11th International Conference on Distributed Smart Cameras*, pages 1–8, 2017.

[108] Hong Chang, Dit-Yan Yeung, and Yimin Xiong. Super-resolution through neighbor embedding. In *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, volume 1, pages I–I. IEEE, 2004.

[109] Ying Chen, Qing Li, Aoyang Zhang, Longhao Zou, Yong Jiang, Zhimin Xu, Junlin Li, and Zhenhui Yuan. 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*, pages 74–81, 2021.

[110] Zhibo Chen, Jianfeng Xu, Yun He, and Junli Zheng. Fast integer-pel and fractional-pel motion estimation for h. 264/avc. *Journal of visual communication and image representation*, 17(2):264–290, 2006.

[111] Ka Leong Cheng, Yueqi Xie, and Qifeng Chen. Iicnet: A generic framework for reversible image conversion. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1991–2000, 2021.

[112] Zhengxue Cheng, Heming Sun, Masaru Takeuchi, and Jiro Katto. Learned image compression with discretized gaussian mixture likelihoods and attention modules. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[113] Cisco. Cisco visual networking index: Forecast and methodology, 2016–2021. http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.pdf, July 2017.

[114] Daniel Crankshaw, Xin Wang, Guilio Zhou, Michael J Franklin, Joseph E Gonzalez, and Ion Stoica. Clipper: A {Low-Latency} online prediction serving system. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 613–627, 2017.

[115] M. Dasari, A. Bhattacharya, S. Vargas, P. Sahu, A. Balasubramanian, and S. R. Das. Streaming 360◦ videos using super-resolution. In *Proceedings of the IEEE International Conference on Computer Communications (INFOCOM)*, April 2020.

[116] Mallesham Dasari, Kumara Kahatapitiya, Samir R. Das, Aruna Balasubramanian, and Dimitris Samaras. Swift: Adaptive video streaming with layered neural codecs. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 103–118, Renton, WA, April 2022. USENIX Association.

[117] Florin Dobrian, Vyas Sekar, Asad Awan, Ion Stoica, Dilip Joseph, Aditya Ganjam, Jibin Zhan, and Hui Zhang. Understanding the impact of video quality on user engagement. In *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM)*, 2011.

[118] Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. Image super-resolution using deep convolutional networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(2):295–307, 2016.

[119] Chao Dong, Chen Change Loy, and Xiaoou Tang. Accelerating the super-resolution convolutional neural network. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Computer Vision – ECCV 2016*, pages 391–407, Cham, 2016. Springer International Publishing.

[120] K. Fatahalian. The rise of mobile visual computing systems. *IEEE Pervasive Computing*, 15(2):8–13, Apr 2016.

[121] F. H. P. Fitzek and M. Reisslein. Mpeg-4 and h.263 video traces for network performance evaluation. *IEEE Network*, 15(6):40–54, Nov 2001.

[122] Sadjad Fouladi, John Emmons, Emre Orbay, Catherine Wu, Riad S Wahby, and Keith Winstein. 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)*, pages 267–282, 2018.

[123] Sadjad Fouladi, Riad S. Wahby, Brennan Shacklett, Karthikeyan Vasuki Balasubramaniam, William Zeng, Rahul Bhalerao, Anirudh Sivaraman, George Porter, and Keith Winstein. Encoding, fast and slow: Low-latency video processing using thousands of tiny threads. In *Proceedings of the USENIX Conference on Networked Systems Design and Implementation (NSDI)*, pages 363–376, Berkeley, CA, USA, 2017. USENIX Association.

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

[125] Aditya Ganjam, Faisal Siddiqui, Jibin Zhan, Xi Liu, Ion Stoica, Junchen Jiang, Vyas Sekar, and Hui Zhang. C3: Internet-scale control plane for video quality optimization. In *Proceedings of the USENIX Conference on Networked Systems Design and Implementation (NSDI)*, volume 15, pages 131–144, 2015.

[126] Yu Guan, Chengyuan Zheng, Xinggong Zhang, Zongming Guo, and Junchen Jiang. Pano: Optimizing 360° video streaming with a better understanding of quality perception. In *Proceedings of the ACM Special Interest Group on Data Communication*, SIGCOMM '19, page 394–407, New York, NY, USA, 2019. Association for Computing Machinery.

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

[128] Onur Gonen Guleryuz, Phil Chou, Hugues Hoppe, Danhang "Danny" Tang, Ruofei Du, Philip Davidson, and Sean Fanello. Sandwiched image compression: Wrapping neural networks around a standard codec. In *2021 IEEE International Conference on Image Processing (ICIP)*, pages 3757–3761, Anchorage, Alaska, 2021.

[129] Onur Gonen Guleryuz, Phil Chou, Hugues Hoppe, Danhang "Danny" Tang, Ruofei Du, Philip Davidson, and Sean Fanello. Sandwiched image compression: Increasing the resolution and dynamic range of standard codecs. In *2022 Picture Coding Symposium (PCS)*, 2022. Best Paper Finalist.

[130] Yulan Guo, Hanyun Wang, Qingyong Hu, Hao Liu, Li Liu, and Mohammed Bennamoun. Deep learning for 3d point clouds: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 43(12):4338–4364, 2020.

[131] Fred Halsall. *Multimedia Communications*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 2000.

[132] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 1026–1034, 2015.

[133] Alain Hore and Djemel Ziou. Image quality metrics: Psnr vs. ssim. In *Pattern recognition (icpr), 2010 20th international conference on*, pages 2366–2369. IEEE, 2010.

[134] Daniel Reiter Horn, Ken Elkabany, Chris Lesniewski-Laas, and Keith Winstein. The design, implementation, and deployment of a system to transparently compress hundreds of petabytes of image files for a file-storage service. In *NSDI*, pages 1–15, 2017.

[135] Hanzhang Hu, Debadeepta Dey, Martial Hebert, and J Andrew Bagnell. Anytime neural network: a versatile trade-off between computation and accuracy. *arXiv preprint arXiv:1708.06832*, 2018.

[136] Pan Hu, Junha Im, Zain Asgar, and Sachin Katti. Starfish: resilient image compression for aiot cameras. In *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*, pages 395–408, 2020.

[137] Pan Hu, Rakesh Misra, and Sachin Katti. Dejavu: Enhancing videoconferencing with prior knowledge. In *Proceedings of the 20th International Workshop on Mobile Computing Systems and Applications*, pages 63–68. ACM, 2019.

[138] Yueyu Hu, Wenhan Yang, and Jiaying Liu. Coarse-to-fine hyper-prior modeling for learned image compression. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(07):11013–11020, Apr. 2020.

[139] Qi Huang, Petchean Ang, Peter Knowles, Tomasz Nykiel, Iaroslav Tverdokhlib, Amit Yajurvedi, Paul Dapolito, IV, Xifan Yan, Maxim Bykov, Chuen Liang, Mohit Talwar, Abhishek Mathur, Sachin Kulkarni, Matthew Burke, and Wyatt Lloyd. Sve: Distributed video processing at facebook scale. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, pages 87–103, New York, NY, USA, 2017. ACM.

[140] Sandy Huang, Nicolas Papernot, Ian Goodfellow, Yan Duan, and Pieter Abbeel. Adversarial attacks on neural network policies. *arXiv preprint arXiv:1702.02284*, 2017.

[141] Te-Yuan Huang, Nikhil Handigol, Brandon Heller, Nick McKeown, and Ramesh Johari. Confused, timid, and unstable: Picking a video streaming rate is hard. In *Proceedings of the Internet Measurement Conference (IMC)*, pages 225–238, 2012.

[142] Zheng Hui, Xinbo Gao, Yunchu Yang, and Xiumei Wang. Lightweight image super-resolution with information multi-distillation network. In *Proceedings of the 27th acm international conference on multimedia*, pages 2024–2032, 2019.

[143] Loc N Huynh, Youngki Lee, and Rajesh Krishna Balan. 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*, pages 82–95, 2017.

[144] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[145] Berivan Isik, Onur G Guleryuz, Danhang Tang, Jonathan Taylor, and Philip A Chou. Sandwiched video compression: Efficiently extending the reach of standard codecs with neural wrappers. *arXiv preprint arXiv:2303.11473*, 2023.

[146] Junchen Jiang, Vyas Sekar, Henry Milner, Davis Shepherd, Ion Stoica, and Hui Zhang. Cfa: A practical prediction system for video qoe optimization. In *Proceedings of the USENIX Conference on Networked Systems Design and Implementation (NSDI)*, pages 137–150, 2016.

[147] Junchen Jiang, Vyas Sekar, and Hui Zhang. Improving fairness, efficiency, and stability in http-based adaptive video streaming with festive. In *Proceedings of the International Conference on emerging Networking EXperiments and Technologies (CoNEXT)*, 2012.

[148] Junchen Jiang, Vyas Sekar, and Hui Zhang. Improving fairness, efficiency, and stability in http-based adaptive video streaming with festive. *IEEE/ACM Transactions on Networking (ToN)*, 22(1):326–340, 2014.

[149] Junchen Jiang, Shijie Sun, Vyas Sekar, and Hui Zhang. 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)*, volume 1, page 3, 2017.

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

[151] Soowon Kang, Hyeonwoo Choi, Sooyoung Park, Chunjong Park, Jemin Lee, Uichin Lee, and Sung-Ju Lee. Fire in your hands: Understanding thermal behavior of smartphones. In *The 25th Annual International Conference on Mobile Computing and Networking*, pages 1–16, 2019.

[152] Jaehong Kim, Youngmok Jung, Hyunho Yeo, Juncheol Ye, and Dongsu Han. Neural-enhanced live streaming: Improving live video ingest via online learning. In *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM)*, 2020.

[153] Jaehong Kim, Youngmok Jung, Hyunho Yeo, Juncheol Ye, and Dongsu Han. Neural-enhanced live streaming: Improving live video ingest via online learning. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, pages 107–125, 2020.

[154] Jiwon Kim, Jung Kwon Lee, and Kyoung Mu Lee. Accurate image super-resolution using very deep convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1646–1654, 2016.

[155] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[156] X. Kong, H. Zhao, Y. Qiao, and C. Dong. Classsr: A general framework to accelerate super-resolution networks by data characteristic. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12011–12020, Los Alamitos, CA, USA, jun 2021. IEEE Computer Society.

[157] S Shunmuga Krishnan and Ramesh K Sitaraman. Video stream quality impacts viewer behavior: inferring causality using quasi-experimental designs. *IEEE/ACM Transactions on Networking (ToN)*, 21(6):2001–2014, 2013.

[158] JC Lawrence and JP Bull. Thermal conditions which cause skin burns. *Engineering in Medicine*, 5(3):61–63, 1976.

[159] Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, et al. Photo-realistic single image super-resolution using a generative adversarial network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4681–4690, 2017.

[160] Hankook Lee and Jinwoo Shin. Anytime neural prediction via slicing networks vertically, 2018.

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

[162] Jinyang Li, Zhenyu Li, Ri Lu, Kai Xiao, Songlin Li, Jufeng Chen, Jingyu Yang, Chunli Zong, Aiyun Chen, Qinghua Wu, et al. Livenet: a low-latency video transport network for large-scale live streaming. In *Proceedings of the ACM SIGCOMM 2022 Conference*, pages 812–825, 2022.

[163] Bee Lim, Sanghyun Son, Heewon Kim, Seungjun Nah, and Kyoung Mu Lee. Enhanced deep residual networks for single image super-resolution. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, 2017.

[164] Chaoyi Lin, Jiabao Yao, Fangdong Chen, and Li Wang. A spatial rnn codec for end-to-end image compression. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 13266–13274, 2020.

[165] Xianshang Lin, Yunfei Ma, Junshao Zhang, Yao Cui, Jing Li, Shi Bai, Ziyue Zhang, Dennis Cai, Hongqiang Harry Liu, and Ming Zhang. Gso-simulcast: global stream orchestration in simulcast video conferencing systems. In *Proceedings of the ACM SIGCOMM 2022 Conference*, pages 826–839, 2022.

[166] Hongqiang Harry Liu, Ye Wang, Yang Richard Yang, Hao Wang, and Chen Tian. Optimizing cost and performance for content multihoming. In *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM)*, 2012.

[167] Xi Liu, Florin Dobrian, Henry Milner, Junchen Jiang, Vyas Sekar, Ion Stoica, and Hui Zhang. A case for a coordinated internet video control plane. In *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM)*, pages 359–370, 2012.

[168] Zhenxiao Luo, Zelong Wang, Jinyu Chen, Miao Hu, Yipeng Zhou, Tom ZJ Fu, and Di Wu. 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*, pages 90–97, 2021.

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

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

[171] Ilias Marinos, Robert NM Watson, Mark Handley, and Randall R Stewart. Disk— crypt— net: rethinking the stack for high-performance video streaming. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 211–224, 2017.

[172] Fabian Mentzer, George Toderici, Michael Tschannen, and Eirikur Agustsson. High-fidelity generative image compression. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS'20, Red Hook, NY, USA, 2020. Curran Associates Inc.

[173] David Minnen, Johannes Ballé, and George Toderici. Joint autoregressive and hierarchical priors for learned image compression. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 10794–10803, Red Hook, NY, USA, 2018. Curran Associates Inc.

[174] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 1928–1937, 2016.

[175] Matthew K. Mukerjee, David Naylor, Junchen Jiang, Dongsu Han, Srinivasan Seshan, and Hui Zhang. Practical, real-time centralized control for cdn-based live video delivery. In *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM)*, pages 311–324, 2015.

[176] Ravi Netravali, Anirudh Sivaraman, Somak Das, Ameesh Goyal, Keith Winstein, James Mickens, and Hari Balakrishnan. Mahimahi: Accurate record-and-replay for http. In *Proceedings of the USENIX Annual Technical Conference (ATC)*, pages 417–429, 2015.

[177] Simon Niklaus, Long Mai, and Feng Liu. Video frame interpolation via adaptive convolution. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[178] Erik Nygren, Ramesh K. Sitaraman, and Jennifer Sun. The akamai network: A platform for high-performance internet applications. *ACM Special Interest Group on Operating Systems (SIGOPS) Operating Systems Review (OSR)*, 44(3):2–19, August 2010.

[179] Junghun Oh, Heewon Kim, Seungjun Nah, Cheeun Hong, Jonghyun Choi, and Kyoung Mu Lee. Attentive fine-grained structured sparsity for image restoration. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2022.

[180] Ooyala. Ooyala global video index q1 2017. http://go.ooyala.com/rs/447-EQK-225/images/Ooyala-Global-Video-Index-Q1-2017.pdf. Last accessed: July 2017.

[181] Zhihong Pan, Baopu Li, Dongliang He, Mingde Yao, Wenhao Wu, Tianwei Lin, Xin Li, and Errui Ding. Towards bidirectional arbitrary image rescaling: Joint optimization and cycle idempotence. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 17389–17398, 2022.

[182] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *Proceedings of the IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2016.

[183] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *Proceedings of the Advances in Neural Information Processing Systems (NIPS) Workshop*, 2017.

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

[185] Haakon Riiser, Paul Vigmostad, Carsten Griwodz, and Pål Halvorsen. Commute path bandwidth traces from 3g networks: analysis and applications. In *Proceedings of the ACM Multimedia Systems Conference (MMSys)*, pages 114–118, 2013.

[186] Francisco Romero, Qian Li, Neeraja J Yadwadkar, and Christos Kozyrakis. {INFaaS}: Automated model-less inference serving. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, pages 397–411, 2021.

[187] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533, 1986.

[188] Sandvine. 2016 global internet phenomena report: North america and latin america. https://www.sandvine.com/downloads/general/global-internet-phenomena/2016/global-internet-phenomena-report-latin-america-and-north-america.pdf. Last accessed: July 2017.

[189] Samuel Schulter, Christian Leistner, and Horst Bischof. Fast and accurate image upscaling with super-resolution forests. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3791–3799, 2015.

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

[191] Wenzhe Shi, Jose Caballero, Ferenc Huszár, Johannes Totz, Andrew P Aitken, Rob Bishop, Daniel Rueckert, and Zehan Wang. Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1874–1883, 2016.

[192] Wenzhe Shi, Jose Caballero, Christian Ledig, Xiahai Zhuang, Wenjia Bai, Kanwal Bhatia, Antonio M Simoes Monteiro de Marvao, Tim Dawes, Declan O'Regan, and Daniel Rueckert. Cardiac image super-resolution with global correspondence using multi-atlas patchmatch. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 9–16. Springer, 2013.

[193] Kevin Spiteri, Rahul Urgaonkar, and Ramesh K Sitaraman. Bola: Near-optimal bitrate adaptation for online videos. In *Proceedings of the IEEE International Conference on Computer Communications (INFOCOM)*, pages 1–9. IEEE, 2016.

[194] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[195] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press Cambridge, 1998.

[196] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Proceedings of the Advances in Neural Information Processing Systems (NIPS)*, pages 1057–1063, 2000.

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

[198] Linpeng Tang, Qi Huang, Amit Puntambekar, Ymir Vigfusson, Wyatt Lloyd, and Kai Li. Popularity prediction of facebook videos for higher quality streaming. In *Proceedings of the USENIX Annual Technical Conference (ATC)*, 2017.

[199] Lucas Theis, Wenzhe Shi, Andrew Cunningham, and Ferenc Huszár. Lossy image compression with compressive autoencoders. In *International Conference on Learning Representations*, 2017.

[200] George Toderici, Damien Vincent, Nick Johnston, Sung Jin Hwang, David Minnen, Joel Shor, and Michele Covell. Full resolution image compression with recurrent neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[201] Yiding Wang, Weiyan Wang, Duowen Liu, Xin Jin, Junchen Jiang, and Kai Chen. Enabling edge-cloud video analytics for robotics applications. In *Proceedings of the IEEE International Conference on Computer Communications, Virtual Conference*, pages 10–13, 2021.

[202] Yiding Wang, Weiyan Wang, Junxue Zhang, Junchen Jiang, and Kai Chen. 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, page 18, USA, 2019. USENIX Association.

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

[204] John Watkinson. *The MPEG Handbook: MPEG-1, MPEG-2, MPEG-4*. Taylor & Francis, 2004.

[205] Patrick Wendell, Joe Wenjie Jiang, Michael J. Freedman, and Jennifer Rexford. Donar: Decentralized server selection for cloud services. In *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM)*, 2010.

[206] David H Wolpert. The lack of a priori distinctions between learning algorithms. *Neural Computation*, 8(7):1341–1390, 1996.

[207] Mingqing Xiao, Shuxin Zheng, Chang Liu, Yaolong Wang, Di He, Guolin Ke, Jiang Bian, Zhouchen Lin, and Tie-Yan Liu. Invertible image rescaling. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part I 16*, pages 126–144. Springer, 2020.

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

[209] Francis Y Yan, Hudson Ayers, Chenzhi Zhu, Sadjad Fouladi, James Hong, Keyi Zhang, Philip Alexander Levis, and Keith Winstein. Learning in situ: a randomized experiment in video streaming. In *NSDI*, volume 20, pages 495–511, 2020.

[210] Hyunho Yeo, Chan Ju Chong, Youngmok Jung, Juncheol Ye, and Dongsu Han. Nemo: enabling neural-enhanced video streaming on commodity mobile devices. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*, pages 1–14, 2020.

[211] Hyunho Yeo, Sunghyun Do, and Dongsu Han. How will deep learning change internet video delivery? In *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*, pages 57–64. ACM, 2017.

[212] Hyunho Yeo, Youngmok Jung, Jaehong Kim, Jinwoo Shin, and Dongsu Han. Neural adaptive content-aware internet video delivery. In *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, pages 645–661, 2018.

[213] Hyunho Yeo, Hwijoon Lim, Jaehong Kim, Youngmok Jung, Juncheol Ye, and Dongsu Han. Neuroscaler: neural video enhancement at scale. In *Proceedings of the ACM SIGCOMM 2022 Conference*, pages 795–811, 2022.

[214] Xiaoqi Yin, Abhishek Jindal, Vyas Sekar, and Bruno Sinopoli. A control-theoretic approach for dynamic adaptive video streaming over http. In *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM)*, 2015.

[215] Anlan Zhang, Chendong Wang, Bo Han, and Feng Qian. {YuZu}:{Neural-Enhanced} volumetric video streaming. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 137–154, 2022.

[216] Xu Zhang, Yiyang Ou, Siddhartha Sen, and Junchen Jiang. SENSEI: Aligning video streaming quality with dynamic user sensitivity. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, pages 303–320. USENIX Association, April 2021.

[217] Yulun Zhang, Yapeng Tian, Yu Kong, Bineng Zhong, and Yun Fu. Residual dense network for image super-resolution. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2472–2481, 2018.

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

[219] Hengyuan Zhao, Xiangtao Kong, Jingwen He, Yu Qiao, and Chao Dong. Efficient image super-resolution using pixel attention. In *European Conference on Computer Vision*, pages 56–72. Springer, 2020.

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

[221] Size Zheng, Yun Liang, Shuo Wang, Renze Chen, and Kaiwen Sheng. 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*, pages 859–873, 2020.

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

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

[224] Wilman WW Zou and Pong C Yuen. Very low resolution face recognition problem. *IEEE Transactions on Image Processing*, 21(1):327–340, 2012.