# RECEIVER-DRIVEN VIDEO ADAPTATION

Aaron J Smith

A dissertation submitted to the faculty of the University of North Carolina at Chapel Hill in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Computer Science.

Chapel Hill
2021

Approved by:

Ketan Mayer-Patel

Leonard McMillian

Shahriar Nirjon

Donald Porter

Montek Singh

## ABSTRACT

AARON J SMITH: Receiver-Driven Video Adaptation
(Under the direction of Ketan Mayer-Patel)

In the span of a single generation, video technology has made an incredible impact on daily life. Modern use cases for video are wildly diverse, including teleconferencing, live streaming, virtual reality, home entertainment, social networking, surveillance, body cameras, cloud gaming, and autonomous driving. As these applications continue to grow more sophisticated and heterogeneous, a single representation of video data can no longer satisfy all receivers. Instead, the initial encoding must be adapted to each receiver's unique needs.

Existing adaptation strategies are fundamentally flawed, however, because they discard the video's initial representation and force the content to be re-encoded from scratch. This process is computationally expensive, does not scale well with the number of videos produced, and throws away important information embedded in the initial encoding. Therefore, a compelling need exists for the development of new strategies that can adapt video content without fully re-encoding it. To better support the unique needs of smart receivers, diverse displays, and advanced applications, general-use video systems should produce and offer receivers a more flexible compressed representation that supports top-down adaptation strategies from an original, compressed-domain ground truth.

This dissertation proposes an alternate model for video adaptation that addresses these challenges. The key idea is to treat the initial compressed representation of a video as the ground truth, and allow receivers to drive adaptation by dynamically selecting which subsets of the captured data to receive. In support of this model, three strategies for top-down, receiver-driven adaptation are proposed. First, a novel, content-agnostic entropy coding technique is implemented in which symbols are selectively dropped from an input abstract symbol stream based on their estimated probability distributions to hit a target bit rate. Receivers are able to guide the symbol dropping pro-

cess by supplying the encoder with an appropriate rate controller algorithm that fits their application needs and available bandwidths. Next, a domain-specific adaptation strategy is implemented for H.265/HEVC coded video in which the prediction data from the original source is reused directly in the adapted stream, but the residual data is recomputed as directed by the receiver. By tracking the changes made to the residual, the encoder can compensate for decoder drift to achieve near-optimal rate-distortion performance. Finally, a fully receiver-driven strategy is proposed in which the syntax elements of a pre-coded video are cataloged and exposed directly to clients through an HTTP API. Instead of requesting the entire stream at once, clients identify the exact syntax elements they wish to receive using a carefully designed query language. Although an implementation of this concept is not provided, an initial analysis shows that such a system could save bandwidth and computation when used by certain targeted applications.

## TABLE OF CONTENTS

## LIST OF TABLES

## LIST OF FIGURES

## LIST OF ABBREVIATIONS

| | |
|---|---|
| ABR | Adaptive Bit Rate |
| AMVP | Advanced Motion Vector Prediction |
| API | Application Programming Interface |
| AVC/H.264 | Advanced Video Coding |
| BL | Base Layer |
| CABAC | Context-Adaptive Binary Arithmetic Coding |
| CCD | Charge-Coupled Device |
| CDN | Content Delivery Network |
| CMOS | Complementary Metal-Oxide-Semiconductor |
| CTU | Coding Tree Unit |
| CB | Coding Block |
| CU | Coding Unit |
| DASH | Dynamic Adaptive Streaming over HTTP |
| DCT | Discrete Cosine Transform |
| DPB | Decoded Picture Buffer |
| DPCM | Differential Pulse-Code Modulation |
| DST | Discrete Sine Transform |
| DWT | Discrete Wavelet Transform |
| EL | Enhancement Layer |
| GOP | Group of Pictures |
| HEVC/H.265 | High Efficiency Video Coding |
| HTTP | Hypertext Transfer Protocol |
| IDCT | Inverse Discrete Cosine Transform |
| IDR | Instantaneous Decoder Refresh |
| IoT | Internet of Things |
| JPEG | Joint Photographic Experts Group |

| | |
|---|---|
| LPS | Least Probable Symbol |
| MDC | Multiple Description Coding |
| M-JPEG | Motion JPEG |
| MCU | Miminum Coded Unit |
| MDC | Multiple Description Coding |
| MPD | Media Presentation Description |
| MPS | Most Probable Symbol |
| MPEG | Moving Picture Experts Group |
| MSE | Mean Squared Error |
| NDVC | Network Distributed Video Coding |
| PB | Prediction Block |
| PDF | Probability Density Function |
| POC | Picture Order Count |
| PU | Prediction Unit |
| PSNR | Peak Signal-to-Noise Ratio |
| R-D | Rate-Distortion |
| RGB | Color space with red, green, and blue dimensions |
| ROI | Region of Interest |
| SHVC | Scalabilty Extension of HEVC |
| SNR | Signal-to-Noise Ratio |
| SSIM | Structural Similarity Index Measure |
| SVC | Scalable Video Coding |
| TB | Transform Block |
| TU | Transform Unit |
| VLC | Variable-Length Coding |
| VR | Virtual Reality |
| VVC/H.266 | Versatile Video Coding |

# CHAPTER 1

## Introduction

One of the reasons video compression research is so exciting is because of the myriad ways in which video technology has affected daily life. In the span of a single generation, video has permanently changed the way entire sectors are handled, including communication, business, education, entertainment, advertising, medicine, transportation, surveillance, and others. Video has revolutionized the way people interact with friends, family, colleagues, and strangers. Most recently, during the global COVID-19 pandemic, multimedia shared through the internet has enabled the world to connect even when isolating. It is thrilling to have the opportunity to study and contribute to a technology that has touched and improved the human experience at such a global scale.

In a way, the scale and rapidity with which video technology has crept into daily life over the past few decades has underscored and exacerbated the very question explored by this dissertation: how best to provide compressed-domain video data to heterogeneous receivers at scale. The explosive growth and rapid integration of video technology in society has resulted in myriad diverse use cases for video data. Teleconferencing, live streaming, virtual reality (VR), 360° video, home entertainment, high dynamic range content, 4K content, social networking, surveillance, security, autonomous vehicles, mobile consumption, object tracking, and cloud gaming are just a few examples. At the same time, advances in image sensing technology allow consumer-grade video sensors to capture more data from the environment than ever before. Data compression is simply a requirement to store, transport, and process the massive bit streams produced by these sensors. As a result, video compression has become a tightly integrated component of the diverse applications listed above.

However, as these two opposing trends continue, they expose a fundamental limitation of the de facto system model for video distribution which treats compression as a black box abstraction. On one hand, advances in video sensor technology necessitate tightly integrated compression that is all but built-in to the sensor (i.e. compression must take place early). On the other hand, diverse, sophisticated applications require that the encoding process is delayed until a tailored, unique version can be produced to meet each receiver's individual needs (i.e. compression must take place late). Since compression is treated by the current model as a black box, there is no compromise between these two opposing forces.

Instead, existing solutions rely on transcoding the video content away from its initial representation. Transcoding either takes place in advance (i.e. at or near capture time), or it takes place on-demand (i.e. at or near request time). Both of these options have significant disadvantages. If transcoding takes place at capture time, the encoder must anticipate in advance the correct encoding parameters that will be needed later by the receiver. This is the case with adaptive streaming techniques like MPEG-DASH as well as scalable encoding approaches like SVC or SHVC. Common coding parameters that a receiver might need to customize include target bit rate, bit depth, region of interest, resolution, frame rate, etc. And as receivers and use cases continue to diversify, we can expect that these parameters will be increasingly difficult to predict at encoding time.

Alternatively, if transcoding takes place at request time, the client is able to direct the process by specifying its needs exactly to the encoder as part of the request. But this approach requires performing a full transcode of the original content in response to each client request. Transcoding is a computationally expensive operation, so this approach does not scale well with the number of clients. Thus, on-demand transcoding has not seen much traction in practice.

The tension between these two opposing strategies is quickly becoming apparent in existing video infrastructure. Video distribution organizations like Netflix, YouTube, and Twitch have so far been able to keep up with heterogeneous client requests through systems-based solutions like MPEG-DASH. But these solutions require significant computational resources and storage, do not

scale well for large number of video producers, and are limited in their ability to handle diverse requests. Furthermore, few organizations actually have the resources necessary to implement existing solutions at scale, resulting in monopolization of the video distribution industry.

This thesis proposes an alternate model for video distribution which would eliminate many of the issues induced by the tension described above. The key to the proposed model is to treat the initial compressed representation of a video that was originally produced by the capture device as the ground truth representation. To support heterogeneous requests for this data, the original compressed representation should be adapted on demand to match the request, while maintaining as much of the initial structure as possible. We call this approach *top-down adaptation*. Existing systems handle adaptation by full transcoding, which is computationally expensive and is not faithful to the original representation. Instead, the proposed approach relies on smart clients to be able to understand their own needs and priorities when making requests for the video data. Clients therefore are given the opportunity to specify precisely which subset of a video's encoded content is needed to achieve their application-level goals. The requested data is extracted and transmitted in a way that retains the same structure as the original source encoding. This approach has dual benefits of reducing per-request computational expense while also ensuring that the client receives data that most closely reflects the ground truth initial encoding.

In support of these ideals, two creative new strategies for achieving top-down adaptation are proposed, implemented, and demonstrated. Both approaches consider the initial representation to be the ground truth, and work by adapting from this initial representation downward to meet the receiver's requested bit rate.

First, a fundamental, content-agnostic technique for top-down adaptation is proposed in which adaptation is characterized as a symbol-dropping problem during the entropy coding process. Since the proposed approach is applied at the entropy layer, it could be used to adapt any content as long as it is represented as an abstract symbol stream. The key idea is for a deterministic, symbol-wise rate controller algorithm to be shared across the entropy encoder and decoder. For each source-coded symbol, the rate controller determines whether it will be encoded in the bit

stream. A value is included in the bit stream only if the least probable symbol could hypothetically be encoded without exceeding the rate controller's target bit rate. The symbols that are not included are *dropped* from the bit stream, and may be sent to an optional, dependent side channel or other layer instead. The symbol dropping process is replicated at both the encoder and decoder, so both know whether each symbol is included. The result is a receiver-driven, adaptive encoding of the initial symbol stream. To assess the feasibility of the approach with video data, entropy-based adaptation is applied both to an H.265/HEVC bit stream and to a M-JPEG bit stream. The results show that top-down entropy-based adaptation works well with non-predictive codecs like M-JPEG. Conversely, content encoded with H.265/HEVC is highly predictive. Experimental results indicate that standard-compliant H.265/HEVC symbol streams are too interdependent to support receiver-driven adaptation. Thus, this work suggests that there is room for a new predictive video codec that is more amenable to top-down adaptation. Development of such a codec is left to future work.

Despite the demonstrated need for receiver-driven adaptation, most video content will continue to be encoded using highly predictive codecs for the foreseeable future. Thus, the second section of this dissertation explores a new technique for supporting receiver-driven adaptation when the initial encoding is highly predictive. In particular, a fast transcoding approach is proposed where all prediction syntax elements are forwarded unaltered to the receiver, and the residual signal is requantized to meet the target bit rate. While such an approach would normally accrue decoder drift due to desynchronized decoded picture buffers, a drift compensation mechanism is implemented at the transcoder which tracks the drift experienced by the decoder and adds the computed drift error to the residual before requantization. The result is a fast transcoder where the encoded bit rate can be specified dynamically by the receiver. This technique is successfully applied to H.265/HEVC, proving that receiver-driven, adaptive, dynamic, bit-rate scalable compression is possible even with predictive codecs.

Finally, an early-stage idea is explored in which the syntax elements of a pre-encoded video are cataloged and exposed directly to clients through an API. An adaptive server decodes the syntax elements from the source bit stream and stores them in an intermediate representation on

disk. Next, the syntax elements are addressed and exposed through an API. Instead of requesting the entire stream at once, clients select the syntax elements that they wish to receive through a carefully designed query language. To save bandwidth and avoid unnecessary computation, clients can identify and request only the syntax elements that are useful to them. For example, clients may choose to request syntax elements based on their semantic meaning within the stream, region of interest in the video, timing, etc. Although an implementation of this system is left to future work, a strategy for addressing syntax elements is proposed using a simplified custom toy hybrid prediction-transform codec. A basic analysis of the example compression system shows that bandwidth and computation can be saved when applied to certain targeted use cases.

The remainder of this chapter introduces and outlines the themes and motivations for the work that constitutes this dissertation. First, evidence is presented to support the claimed opposing trends of receiver diversification and tightly integrated compression. Next, the existing solutions to these challenges are described in terms of the de facto black-box model of compression, with special attention to the limitations of these methods. In contrast, a proposed alternate model is offered which treats the original compressed representation of a video as the ground truth, and adapts downward based on the requests of smart clients capable of understanding and articulating their video needs. The proposed solutions are described in the context of this new model, culminating in a thesis statement for the dissertation. Finally, an outline is presented for the remainder of the dissertation.

## 1.1 Motivation

This dissertation identifies two video coding trends which have become apparent over the past few years. These two opposing trends form the motivation for the proposed work. First, video formats and use cases are rapidly diversifying, resulting in video clients that require their own unique encoding of a given captured piece of content. Second, video sensor technology has improved, and consumer-grade sensors now produce so much raw data that on-chip compression circuits have become widespread. These two trends are detailed with evidence below.

### 1.1.1 Trend 1: Diverse Receivers

Every year, Cisco publishes an annual update to their Visual Networking Index, a public white paper well-known in industry and academic circles for providing a sentinel perspective of networking trends worldwide. Each annual update highlights different concerns, achievements, and trends based on latest growth statistics. Industry organizations use these statistics to shape their product offerings and services to meet anticipated consumer demand. In the past few years, Cisco has been using their annual publications intentionally to quantify ways in which internet traffic is currently diversifying. The 2017 and 2021 publications identify the following trends, which focus specifically on video data (Cisco, 2017).

1. **Staggering Growth** – In 2021, one million minutes of video will be transferred over the internet every second, constituting 82% of all IP traffic.

2. **Live Streaming** – While a lot of video traffic, including that from popular video streaming providers like Netflix and Hulu, is prerecorded, the share of video that is produced and streamed live is also experiencing growth; by 2021, Cisco projects that 13% of all video traffic will come from live-streamed sources. The rising popularity of live-streaming services like Facebook Live and Twitch; internet TV streaming services such as SlingTV; and live video conferencing services like FaceTime, Skype, and Google Hangouts have all contributed to make live-streaming an important branch of multimedia networking.

3. **Mobile Receivers** – Internet usage is quickly moving in the direction of mobile devices; in 2021, smartphone-initiated traffic is projected to exceed PC-initiated traffic on the internet. Ultimately, as internet users' insatiable desire for video content and mobile access grows, smartphones will be increasingly relied upon to act as the primary recorders, storage units, transmitters, and players of the video content their owners demand.

4. **Machine-to-Machine Use** – Machine-to-machine (M2M) connections are experiencing rapid growth as interest in internet of things (IoT) and connected home appliances explodes.

Many of these systems use video, including video surveillance, healthcare monitoring, transportation, autonomous driving, and package or asset tracking. By 2023, Cisco projects that M2M connections will represent 50% of devices and connections that use the internet.

Ostensibly, these statistics evidence that the world now consumes *more* video content through *increasingly diverse* devices, network connections, and experiences. Today, internet users realistically expect to receive video in a wide variety of ways, including (1) low-latency live streaming, (2) streaming over mobile or unreliable network connections, (3) playback on high-quality 4K high dynamic range (HDR) displays, (4) live conferencing, and (5) playback on low-powered smartphones or other mobile devices. Additionally, machines also play an active role in consuming internet video through sophisticated vision algorithms that automatically infer information about the captured scene as they "watch" video.

### 1.1.2 Trend 2: Tightly Integrated Compression

As light sensor technology has improved, even cheap, consumer-grade sensors now are able to sense at high resolution and bit depth. Most new smartphones are equipped with sensors that can capture video at 4K resolution or higher. So even as users are capturing more and more minutes of video, the video itself is being produced at higher bit rates. As an example, consider 4K video data captured at 60 frames per second, with 3 color channels and a bit depth of 8. The resulting bit rate for this uncompressed, raw 4K video is

$$8 \times 3 \times 3840 \times 2160 \times 60 = 11.9 \text{ Gbps} \tag{1.1}$$

This represents nearly 1.5 GB for every second of video captured. Not only is this a huge amount of data to store, but the bit rate exceeds the write speed of SATA bus interface technology. Thus, compression is simply a necessity before any captured data can be stored.

To solve this problem, compression must be applied before the video is even stored. This is exactly the approach taken by consumer-grade video sensors, which are equipped with a

hardware-based compression chip that takes as input the raw video data and produces a compressed representation as output. At the time of publication, H.264/AVC is the most common compressed format used by these hardware chips, but H.265/HEVC is gaining market share. The raw video data itself never actually makes it off the sensor chip. Instead, the first representation obtained off the sensor is the compressed-domain bit stream produced in the hardware.

At a high level, the increases in data bit rates that have followed from advances in sensing technology have forced compression to become tightly integrated in all video applications. A video's entire life cycle must be in the compressed domain, from the moment of capture to the instant before it is displayed. Furthermore, any transcoding operation that occurs after the initial encoding degrades the quality of the reproduction. This suggests that the ground truth representation of video data is its initial compressed representation. All other representations—including both transcoded compressed streams and decoded pixel-space representations—can be interpreted as alternate renderings of the original.

### 1.1.3 Growing Tension

The recent trends in video coding system design are in opposition to one other. Advanced use cases require compression that is tightly integrated in the application life cycle. No longer is a single encoded representation of a video suitable for all use cases. As receivers become more sophisticated and diverse, each one needs a different unique subset of the original captured video stream. The naive solution of sending all captured data to every receiver is not feasible, since receivers are typically limited by their available network bandwidth. Thus, separate encodings must be created for different types of receiver, based on which subset of video data is most relevant to their needs. If it is possible to predict the eventual use case for a video's content, this process can be performed in advance, before the requests are made. But as receivers continue to diversify, it becomes more difficult to predict their needs. And some applications, including some VR systems, require a tight dynamic feedback loop between the display device and the source. In extreme cases, extraction and compression must take place on a per-user, per-request basis.

Clearly, advanced receivers are better suited when video compression takes place later, after more information is known about the use case. However, delaying compression until request time—as an advanced receiver would prefer—is not an option either due to the sheer size of raw, uncompressed video. With limited exceptions, video data is almost always immediately compressed by its sensor at capture time. This is because commodity sensor technology has advanced to the point where even cheap sensors produce data at excessively large bandwidths. Writing and storing the uncompressed video data produced by these sensors is impossible with SATA bus technology. Therefore, commodity sensors are designed with integrated on-board hardware compression chips that produce an encoded representation of all captured data. And since this encoded representation is the only one that leaves the chip, it represents the ground truth representation.

Thus, an inevitable tension exists in the current system design for video compression that will only continue to worsen. On one hand, video must be compressed early as a matter of necessity, just to make storage and transfer of the captured data feasible. On the other, diverse receivers are best served when compression takes place late, after the specific application and use case goals are known. Ultimately, the resolution between these two opposing goals is that the initial compressed representation of video content must be *adapted* to suit each application's needs. And as the trends of receiver diversification and tight integration of compression continue into the 2020's, the need for adaptation strategies for video data that are efficient and effective has never been more clear.

## 1.2 Problem Definition

To aid in the presentation of existing strategies for video adaptation, it helps to first clearly define the problem. Therefore, the following steps outline the envisioned life cycle of video content, from the moment it is captured to the time it is displayed. In particular, *adaptation* is defined as an intermediate step whereby the captured content is adapted from its source representation to meet the needs of the client.

1. **Video Source** – Raw, uncompressed, digital video content is originally created by a light sensor, screen capture, 2D or 3D rendering, or other video producer device. The content represents natural video and exhibits both spatial and temporal cohesion.

2. **Initial Compression** – The source device is designed to produce an initial compressed representation of the video, a process that likely takes place in hardware. The compression parameters used to guide creation of the initial representation generally cannot be controlled by the adaptation system.

3. **Adaptation** – At some point, the initial compressed representation is adapted to fit its use case. This may involve transcoding to adjust coding parameters such as bit rate, resolution, bit depth, frame rate, etc. By the time adaptation takes place, at least some knowledge of the video's use case exists so that the parameters can be tuned appropriately. If the use case can be anticipated, adaptation might take place before a request is made.

4. **Heterogeneous Requests** – Either asynchronously or synchronized with the capture time of the video, one or more distributed client devices make a request for the video data. Each client has a different application-level goal for requesting the video and different computing resources available for achieving this goal. Based on the client's profile, they may not need all data originally captured by the source. Depending on the capabilities of the system, a client may be able to express the subset of data that it needs, and the provider may be able to extract and respond with only the requested information. This can conserve network bandwidth as well as computing resources for both the client and the provider.

5. **Smart Clients** – Ultimately, the video data is received by the client, decoded, and used in the application. Traditionally, the primary purpose of video is to be displayed to a human. However, the growth of M2M applications is challenging this assumption. In the future, machine-endpoint applications may become the dominant use case.

## 1.3 Existing Approaches

This section describes the existing strategies for video adaptation. Existing adaptive techniques are characterized by the assumption that data compression is a black-box abstraction, and that the initial encoding of a video is disposable. Adaptation is generally achieved entirely by the system layer instead of within the compression format. These approaches can generally be classified into three categories: scalable coding, adaptive streaming, and full transcoding.

### 1.3.1 Approach 1: Scalable Coding

*Scalable coding* represents the only category of existing approaches in which adaptation is cast as a compression problem. Scalable coding works by encoding video data in a way such that certain designated subsets of the encoded bit stream can be separately decoded to produce an incomplete—but still useful—approximation of the stream. The most common organizational strategy for scalable coding is a *layered* approach, where the encoded data is separated into a *base layer* and a series of *enhancement layers*. The base layer must be received by every decoder, and provides the lowest quality decodable approximation of the stream. The enhancement layers can be decoded and incorporated with the base layer to improve the reconstruction quality. For instance, one layer may provide the necessary information to increase the resolution of the reconstructed frames, while another doubles the frame rate. Decoders can select which layers to receive based on their available resources and application needs. If all layers are received, the full-quality stream can be recovered.

Scalable coding has a long history in video compression research, and has been successfully demonstrated as an extension for many existing codecs including H.264/AVC, H.265/HEVC, and AV1. Despite this, scalable coding has not seen much use in practice. A scalable encoding generally is created for a given video before any client requests are made. The parameters that should be client-scalable therefore must be predicted ahead of time. This means scalable coding approaches are unable to provide fully dynamic, receiver-driven adaptation.

### 1.3.2 Approach 2: Adaptive Streaming

*Adaptive streaming*, also called *simulcast*, is the practice of independently transcoding a single video multiple times, producing a set of independent encodings of the same content. Presumably, each independent encoding is created with a different set of coding parameters, effectively resulting in a set of pre-created options for a client to choose from. All of these options are hosted by a simple file server, alongside a text-based manifest file which simply lists the available encodings. Clients receive the manifest file first, and use it to select and request the encoding that best matches their resources and use case. By segmenting all encodings at synchronized points in time, clients can effectively switch between the different available encodings dynamically, during streaming.

By far, adaptive streaming is the dominant adaptive technique for use in practice. Since requests can be handled by a simple file server, video can be hosted on content delivery networks (CDNs) for fast, distributed delivery that scales well with the number of requests. Like scalable encoding, adaptive streaming requires that the coding parameters needed by clients can be predicted ahead of time.

### 1.3.3 Approach 3: Full Transcoding

*Full transcoding* strategies place a full transcoder in between the content source and the requesting client. Thus, the initial representation produced by the sensor is completely thrown out and replaced with a new encoding according to the client's needs. Full transcoding may take place before a request is made if the client's needs are predictable; or, in fully receiver-driven dynamic situations, transcoding may take place on-demand. On-demand full transcoding places a large per-request computational cost on the consumer, so it does not scale well with the number of requests.

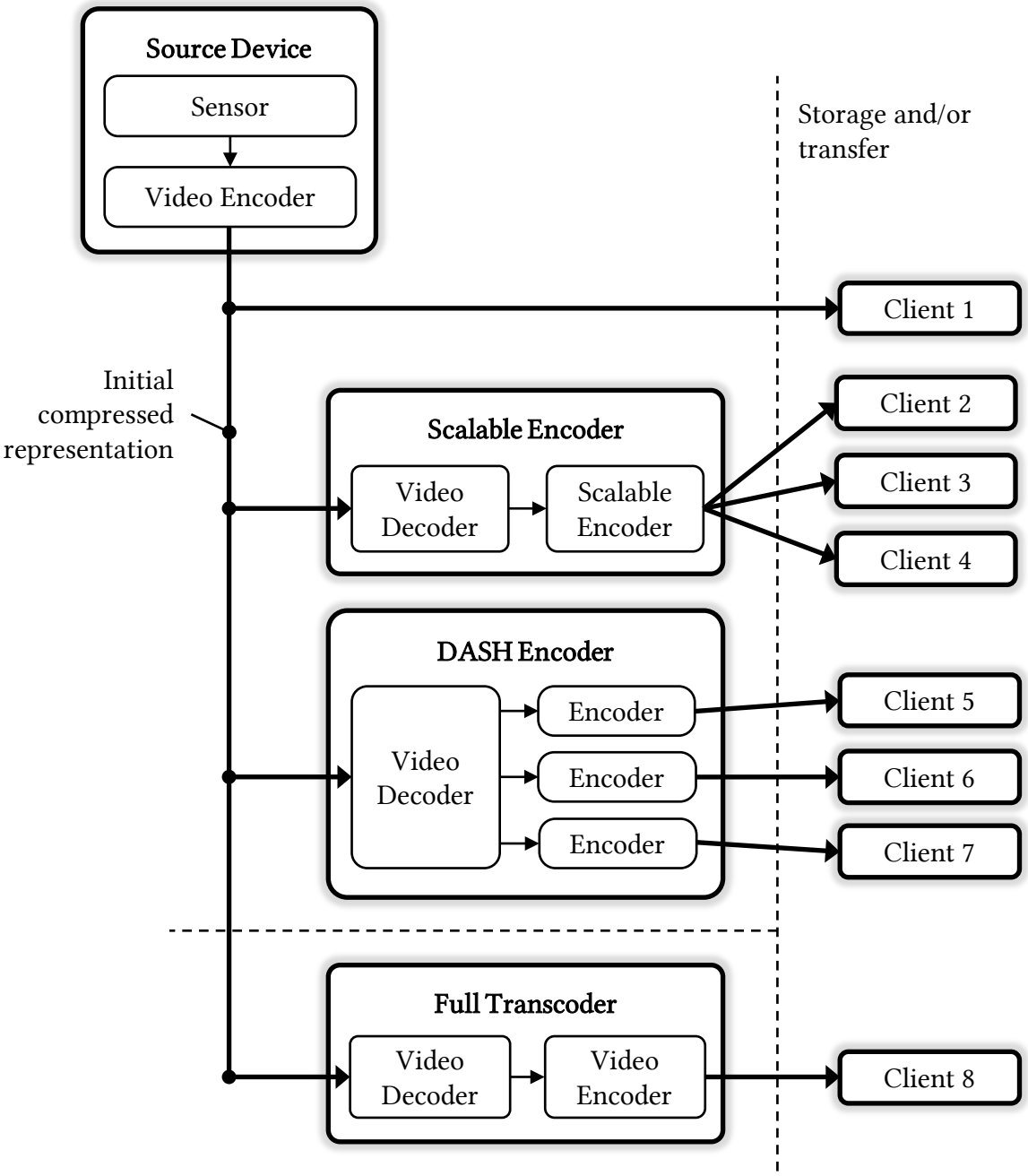



Figure 1.1: This figure depicts the de facto black-box abstraction model for video compression and adaptation. Video data is initially captured and immediately encoded by a sensor. Clients that have the available bandwidth can receive and use this initial encoding. For other clients, the initial encoding must be adapted to their use case and available network bandwidth. Three adaptation strategies are depicted: scalable encoding, simulcast with MPEG-DASH, and full transcoding. None of these adaptive approaches support reuse of the initial representation.

### 1.3.4 System Model Diagram

Figure 1.1 depicts a system model that illustrates all three existing adaptive approaches described above. Visual data is captured by a light sensor and immediately is processed by an accompanying hardware encoder to produce an initial compressed representation bit stream. It is assumed that clients are distributed from the capture device. If no adaptation is needed (or if the client has sufficient bandwidth and can perform adaptation by itself), then the encoded video data can be forwarded directly to the client, as is the case with Client 1. For situations where adaptation is required, the three existing approaches are shown. All three approaches begin by fully decoding the video to pixel space, thereby discarding the initial encoding. Scalable systems support adaptation by generating a scalable encoding of the pixel-space video data in advance of requests. Adaptive streaming (e.g. MPEG-DASH) independently encodes the video multiple times and stores the resulting representations to be streamed by a file server. Full transcoding simply re-encodes the pixel-space data according to the client application's needs.

### 1.3.5 Limitations

The existing approaches to video adaptation are based on a fundamentally misaligned view of the video life cycle, in which the initial compressed representation is disposable and video compression is treated as a black-box abstraction. All three approaches immediately throw out the initial compressed representation, in favor of producing a new representation from pixel space. This is simply an unsustainable approach to adaptation. Recent increases in sensor technology have translated into increases in video bandwidth. To keep up, video codecs have become more complex and tightly inter-dependent so as to maximize compression performance. This has resulted in two effects. First, the initial encoding of a video is even more important than before, since its prediction representation is now more sophisticated and provides semantic information about the captured environment. Second, the process of video encoding is now computationally more taxing than ever before.

These two effects are the nail in the coffin for existing approaches. Existing adaptation strategies dispose of the initial representation and re-encode from scratch. This is inefficient and unsustainable on two levels: (1) it incurs the high computational cost of encoding from scratch, and (2) it throws away the semantically useful information embedded in the initial encoding about the captured environment. The existing approaches will simply always be disadvantaged due to these design choices.

So far, major video distribution organizations including YouTube, Twitch, Netflix, Disney+, Amazon, and others have been able to mask these fundamental systemic flaws through massive investments in computational infrastructure. With enough computing resources, servers can simply pay the price for the inefficiency incurred by re-encoding content using the existing solutions. However, this does not scale well with the number of video producers. As the trends of increasing coding complexity and diverse receivers continue, existing solutions will continue to strain our infrastructure. Furthermore, the high cost of deploying the infrastructure necessary to implement existing solutions will price out smaller video distributors from competing. Already, the video industry is monopolizing around a small number of powerful video distributors who have the resources to keep up.

## 1.4 Proposed Approach

As described in the previous section, the fundamentals of the existing approaches to video adaptation are misaligned with recent trends in video capture and applications. Since these trends show no sign of slowing down, it's time to consider alternate system models that can alleviate the tension that is currently building. This dissertation proposes and evaluates one such model, built upon the following core assumptions about video systems.

- **Receivers are diverse** – As receiver technology continues to advance, receivers are diversifying in many ways. Many of the old assumptions about the capabilities and applications of a video receiver are no longer accurate. Despite this, receivers are unifying around one

commonality: they are becoming smarter and more capable. In the future, receivers will be more aware of their own needs for video content than any encoder is.

- **Source encoding is ground truth** – With the incorporation of on-sensor hardware encoders, a consumer-grade video's life cycle is spent almost entirely as an encoded stream. This suggests that the compressed representation of video is, in fact, the primary representation. The uncompressed, pixel-space representation is merely an alternate rendering of the source data. Furthermore, whenever a video undergoes a transcode operation, the output is farther away from the initial ground truth representation.

- **Applications are sophisticated** – Smart clients of the future may be capable of understanding and using the compressed-domain syntax elements of an encoded video stream to help achieve their diverse application goals. For example, motion vector information is encoded in every video, but is not usually exposed by decoders. If it were available, smart clients could possibly use this data to infer in-scene motion without needing to decode the entire stream.

### 1.4.1 System Model Diagram

Based on these ideals, this dissertation proposes a new system model for video compression and adaptation. The new system treats the source encoding for video data as the ground truth, and implements adaptation as a top-down, receiver-driven process. All adapted versions of the video retain the original semantically valuable prediction information from the initial encoding. Furthermore, semantic scene information like motion vectors can be exposed to smart clients to be used for more advanced application goals.

By treating the initial encoding as ground truth and adapting downward, the proposed model completely avoids the limitations of existing techniques. Full pixel-space transcoding never takes place in the proposed system. This means the computationally expensive encoding process is skipped entirely.

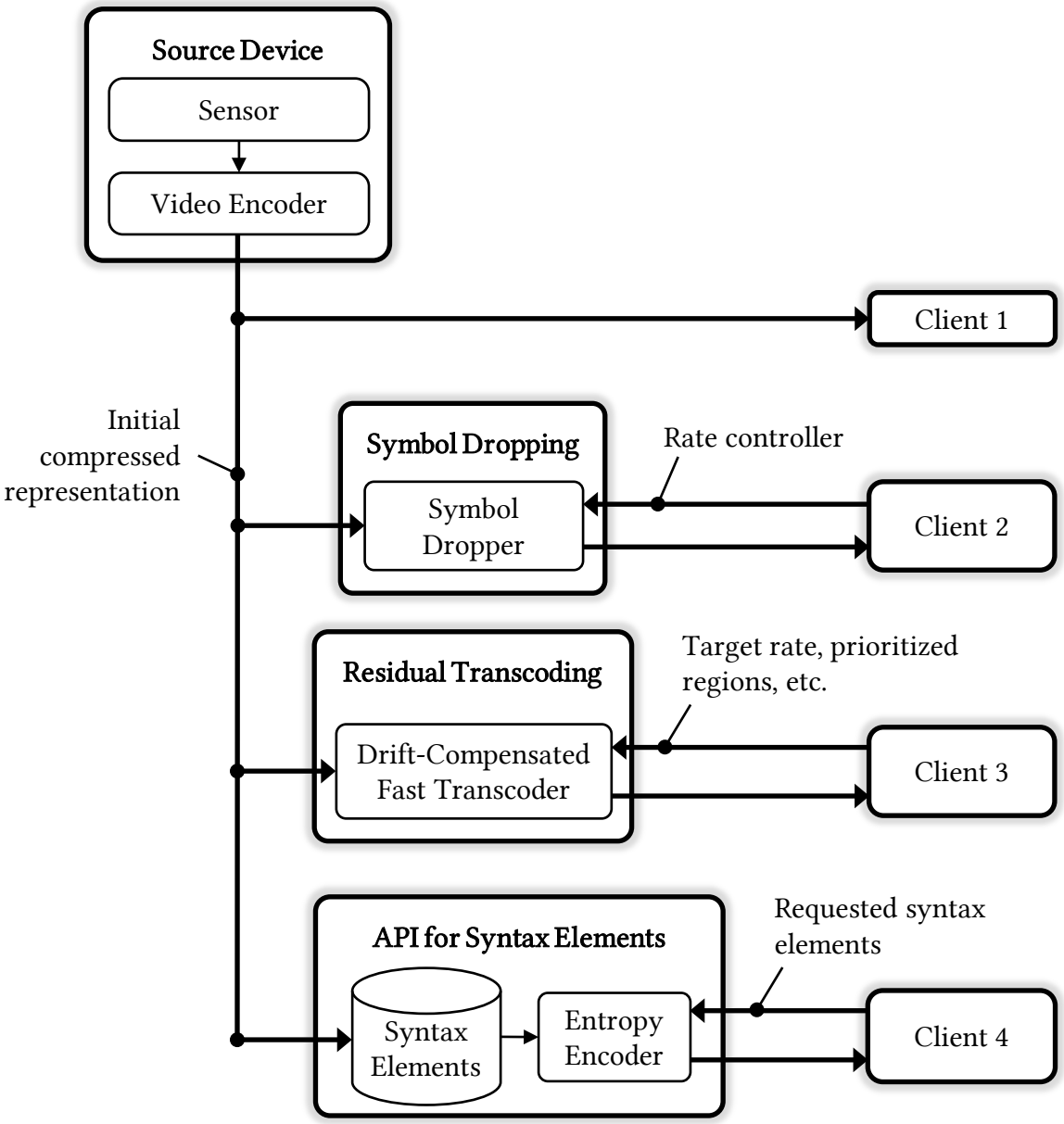



Figure 1.2: This figure depicts the proposed model for video compression and adaptation. Three adaptation strategies are proposed, all of which respect the original compressed representation as the ground truth and work by adapting downward. The first approach treats adaptation as an abstract symbol-dropping problem in the entropy coding layer. The second strategy adapts by requantizing the prediction residual signal while compensating for decoder drift. Finally, the third technique extracts and indexes the originally encoded syntax elements, exposing them to clients through an API.

Figure 1.2 depicts the proposed system model. Three novel, top-down, receiver-driven adaptation strategies are illustrated: entropy-based symbol dropping, drift-free residual transcoding, and a receiver-driven syntax element API. Unlike the existing model depicted in Figure 1.1, none of these proposed adaptation strategies begin by discarding the initial representation. Instead, they adapt downward from the initial encoding in a receiver-driven way. Each of these ideas is briefly described in the next section, along with their intellectual merit.

### 1.4.2 Contributions

In support of the model presented in Figure 1.2, this dissertation contributes the following three unique strategies for achieving top-down, receiver-driven adaptation.

#### Entropy Symbol Dropping

First, I propose a new, fully entropy-based adaptation strategy. The proposed technique uses an abstract symbol stream as its source model, which means it is not specific to video data and could be used to achieve adaptation for any stream of data. The technique revolves around the idea of strategic *symbol dropping*. To achieve adaptation, a synchronized, deterministic rate controller algorithm is reproduced by both the encoder and decoder. For each originally encoded source symbol, this rate controller determines whether that symbol should be included in the adapted stream. This decision is made without knowledge of the symbol's value (since the decoder must be able to follow along), but can incorporate knowledge of the symbol's estimated probability distribution. If the rate controller determines that there is enough bandwidth available to include the symbol (even at its worst-cast cost), then the symbol is encoded; otherwise, it is dropped from the bit stream. Decoders therefore know exactly which symbols are received, and which could not be signaled while still meeting the requested bit rate.

This approach is notable for its flexibility. By allowing the receiver to supply the rate controller algorithm, the process of selecting which symbols to include can be completely receiver-driven. Alternatively, a series of predetermined rate controllers can be applied instead to produce

a scalable, layered encoding of the initial symbols. Either way, adaptation always prioritizes the original encoding with this approach. Receiving more symbols in the adapted bit stream directly corresponds to a closer decoded reconstruction of the original stream. By receiving all symbols, a decoder can reconstruct the original stream exactly. Therefore, the resulting adaptation is top-down and treats the initial encoding as ground truth.

To evaluate the performance of entropy-based adaptation on encoded video data, two experiments are performed. First, an H.265/HEVC bit stream is adapted by decoding the constitute syntax elements from the initial encoding, and then layering these same syntax elements in the adapted version. The results successfully demonstrate that top-down adaptation is possible with video data, as the adaptive rate-distortion curve produced is correctly anchored at the top. However, the tight inter-dependence of H.265/HEVC symbols limits the coding efficiency of this approach. Most symbols, including all prediction and partitioning syntax elements, must be included in the base layer in order for a decoder to even be capable of decoding the rest of the stream. Furthermore, any layered syntax elements not received by a decoder introduce drift in the decoder's picture buffer due to the predictive coding structure used by H.265/HEVC. This drift quickly dominates the recovered signal for decoders that receive only the lowest layers.

These results suggest that the structure of H.265/HEVC is overly inter-dependent, making it a poor candidate for top-down adaptation. In the future, new predictive video codecs could be better designed with top-down adaptation in mind. However, in the meantime, top-down adaptation can be applied to non-predictive video codecs with much better results. This is demonstrated by applying layered, entropy-based adaptation to the symbols of a M-JPEG coded video. M-JPEG streams consist entirely of I-frames and do not use predictive coding. The results clearly show that top-down adaptation works well in a non-predictive environment. The adaptive rate-distortion curve provides a smooth trade-off between rate and distortion. The receiver is able to drive adaptation by indicating exactly which portions of the video in which it is interested. And since the decoder is able to perfectly recover the complete original stream by receiving all layers, this approach successfully treats the source representation as the ground truth as desired.

**Drift-Free Residual Transcoding**

The results of the entropy-based symbol dropping experiment confirm that as currently defined, existing predictive codecs like H.265/HEVC produce a highly inter-dependent symbol stream which is not particularly well-suited to an adaptive approach. Despite this reality, widely distributed hardware encoders all but ensure that most video content will be initially encoded in this way for the foreseeable future. Therefore, it is worthwhile to explore top-down adaptation strategies that are amenable to highly inter-dependent, predictive coding.

To this end, the next major contribution of this dissertation is a drift-compensating residual transcoding technique for H.265/HEVC encoded video. Two salient observations motivate this approach. First, the prediction syntax elements of an H.265/HEVC stream represent the most important semantic data from the ground truth encoding. Thus, this information should be preserved by all adapted versions of the content. Second, decoder drift was the main antagonist of symbol dropping with predictive coding. Therefore, compensating for this drift is crucial to achieve reasonable coding efficiency.

The proposed technique allows a receiver to specify the target bit rate for the adapted stream. For each frame, the original source prediction information is directly re-used by the adapted version. This achieves the goal of the source encoding as ground truth. The residual information is *requantized* by the provider to meet the requested target bit rate. Decoder drift is tracked by the transcoder and compensated during requantization. The resulting system is a fast transcoder which allows the receiver to dynamically drive the exact target bit rate, and operates at per-request complexity similar to that of a decoder.

The drift-compensated residual transcoder is implemented and applied to a test set of H.265/HEVC coded source streams for evaluation. The adaptation rate-distortion curve achieved shows a smooth trade off between rate and distortion that is anchored at the top. At lower bit rates, the adapted stream is unable to maintain the same coding efficiency as a comparable non-adaptive encoding. This is because lower bit rates can sometimes save bits and achieve better compression performance using a less precise prediction. However, signaling a less precise prediction would

undermine the stated goal of retaining the source encoding in all adapted representations. Thus, in our view, the coding efficiency penalty associated with reusing the original prediction is more than offset by the benefits achieved with a top-down approach that treats the initial encoding as ground truth.

### Syntax Element API

Finally, the last contribution of this dissertation is a prospective system in which the encoded syntax elements from a video's source representation are cataloged and exposed by a provider through a well-defined API. Such a system would allow for truly receiver-driven adaptation, where receivers identify and request the exact syntax elements—or groups of syntax elements—that they need. For example, a motion-sensing M2M application could request only the motion vectors of an encoded video, and use them to infer whether significant object motion is present in the scene at each frame. This could result in massive bandwidth and complexity savings for targeted use cases.

This dissertation does not implement or evaluate such an approach. Instead, its philosophical merits are discussed and certain specific use cases that could potentially benefit from such an approach are outlined.

## 1.5 Thesis Statement

As digital video sensors improve, the massive amount of data they produce necessitates tightly integrated data compression. At the same time, video applications and receivers are diversifying and becoming smarter. *To better support the unique needs of smart receivers, diverse displays, and advanced applications, general-use video systems should produce and offer receivers a more flexible compressed representation that supports top-down adaptation strategies from an original, compressed-domain ground truth.*

## 1.6 Dissertation Overview

The remainder of this dissertation is organized into the following chapters.

- **Chapter 2** describes the technical innovations of the late twentieth century which resulted in the hybrid prediction-transform video coding structure. These advances propelled digital video to become the dominant medium for visual technology, and remain integral to all modern video compression formats today.

- **Chapter 3** outlines the *heterogeneous client problem*, which refers to the fact that capture devices, displays, and use cases for video are rapidly diversifying. It makes the case that diversification is causing strain on existing compression resources. Finally, it outlines existing strategies for adapting pre-encoded video to the needs of receivers.

- **Chapter 4** describes the fundamental challenges associated with developing a scalable, predictive codec. It highlights the importance of tracking the received content available for reference at the decoder, and explains how various associated challenges can be mitigated even in a receiver-driven environment.

- **Chapter 5** offers a novel, fundamental strategy for content adaptation that integrates with arithmetic entropy coding. The resulting approach is content-agnostic and could be used to adapt any data source coded as an abstract stream of symbols. To evaluate the system on both predictive and non-predictive codecs, it is applied to both H.265/HEVC and M-JPEG coded streams.

- **Chapter 6** takes a different approach toward solving the heterogeneous client problem, in which pre-encoded video content is adapted on-the-fly in response to each client's request. The proposed technique accounts for decoder drift, ensuring that the reconstruction recovered by the decoder exhibits near-optimal rate-distortion performance.

- **Chapter 7** describes future work: the proposed development of a fully receiver-driven compression system in which a video's constituent syntax elements are exposed through

a HTTP API. Receivers can request the exact syntax elements they need through an expressive syntax element query language.

- **Chapter 8** concludes the dissertation by reviewing the proposed techniques and highlighting the potential impact of the completed efforts.

# CHAPTER 2

# Hybrid Video Coding

A central claim of this thesis is that video systems can provide clients with increased efficiency and flexibility by better leveraging the underlying representation of the data recorded at the video source, instead of treating video compression as a black box. All modern video standards generally employ a similar "hybrid" model for representing and compressing video data that combines multiple independent techniques taken from the image compression, signal processing, and data representation communities (Chen et al., 2001). This section presents a historical overview of the most important elements of the digital video model, with emphasis on areas of particular significance to later chapters.

## 2.1 The Bandwidth Challenge

In the early days of digital video systems research during the mid-twentieth century, a major challenge was the massive amount of data required to represent a digital video signal. Uncompressed, raw video data is well-known to take up incredible amounts of bandwidth; for perspective, a 1920x1080 high-definition video camera that samples incoming light at 30 frames per second (fps) with 8 bits of resolution per color channel produces a data stream of 1.49 gigabits per second (Gbps) of video. At this order of magnitude, raw digital video is simply an impractical technology for all but extremely limited use cases.

For most of the twentieth century, the bandwidth challenge limited the feasibility of digital video, and prevented it from overtaking analog video as the dominant content technology. Given the available technology, analog was simply a more efficient format than digital, both for video storage and transfer.

| | Resolution | Color Depth | Refresh Rate | Data Rate |
|---|---|---|---|---|
| **480p** | 640 x 480 | 24 bits | 30 fps | 221 Mbps |
| **720p** | 1280 x 720 | 24 bits | 30 fps | 663 Mbps |
| **1080p** | 1920 x 1080 | 24 bits | 30 fps | 1.49 Gbps |
| | | | 60 fps | 2.98 Gbps |
| **4K** | 3840 x 2160 | 24 bits | 30 fps | 5.97 Gbps |
| | | | 60 fps | 11.9 Gbps |

Table 2.1: Data rates for selected uncompressed video formats

Things began to change in the 1970's with the development of the hybrid prediction-transform model for representing digital video (Chen et al., 2001). This huge leap forward was the combination of two key technical innovations for video compression: motion-compensated prediction and the discrete cosine transform. Together, they form the hybrid prediction-transform model for digital video which has been the basis for all widespread digital video formats ever since (Fang et al., 2006).

Armed with motion prediction and the discrete cosine transform, video compression was well-equipped to solve the bandwidth problem by the 1990's, propelling digital video to become the dominant technology ever since. These two crucial contributions are described in detail below with important historical context.

## 2.2 Transform Coding

One of the major technical achievements that helped solve the bandwidth problem and enable the digital revolution was the development of the discrete cosine transform for decorrelating visual data into its frequency components. The discrete cosine transform is an example of transform coding, a class of compression techniques where a relatively simple, reversible transformation is applied to raw input data before any compression is applied. The high-level goal of applying a transform is to map the source data into a different—yet equivalent—space where the most relevant qualities of the signal have been decorrelated into independent dimensions. Transform coding is most often applied to digital signals that produce numerical values which can be easily transformed

by an invertible mathematical function. Common domains for this family of techniques include digital audio, image, and video.

Since the transform function is generally reversible, it is a lossless operation that imparts no compression on the data by itself. Instead, its sole purpose is to alter the representation of the source signal so that more and less important features are isolated into separate dimensions. Selecting a suitable transform to exhibit this behavior obviously requires substantial domain knowledge about the particular signal being compressed, as well as an understanding of the end goal of the data. In other words, transform coding is only applicable when one knows (1) how the signal will ultimately be used, and (2) which dimensions of the data contribute most to the success of that goal.

Once an appropriate transform has isolated the important features as defined by the application, a lossy compression algorithm can be applied to the data in transformed, decorrelated space. The lossy process makes strategic decisions about how much to approximate the transformed features in the final representation, taking into account target compression goals as well as how much each feature contributes to the overall application goal (i.e., how "important" each feature is). Coarser approximations of feature values yield higher compression and lower fidelity, while finer approximations result in lower compression but a more faithful reproduction.

The end result of applying lossy compression is an approximated representation of the transformed signal. Any loss incurred by the lossy step is unrecoverable, since the approximated features in transformed space will likely no longer map to their original values. However, since the approximation is applied to the decorrelated signal, it can be targeted to the features that have the least effect on the end goal of the system.

Modern digital video systems generally employ transform coding to apply controlled loss at two strategic points in the overall compression system. First, the discrete cosine transform (DCT) is used to map still, two-dimensional image data into frequency space, where quantization can then be applied to target loss at particular frequencies. Second, a color space transform maps RGB pixel data measured at the source into the YCrCb color space, where subsampling can target loss in the

chromatic dimensions of the signal. Below, these four techniques—DCT, quantization, color space transforms, and subsampling—are described in greater detail with historical context.

### 2.2.1 Discrete Cosine Transform

The discrete cosine transform (DCT) is a reversible mathematical operation such that, when applied to 2D still visual data like an image or video frame, decorrelates the data by frequency—that is, it separates high frequency texture information from low frequency texture information. Through qualitative analysis, researchers discovered that the human visual system (HVS) tends to overemphasize low frequency textures when assessing overall quality of visual data (Winkler et al., 2001). In other words, eliminating the high frequency textures of a visual signal results in the lowest overall reduction of perceived quality when rated by a human observer, in comparison to eliminating all frequencies equally or only eliminating low frequencies.

These engineers also recognized that the human tendency to disregard high frequency textures can be exploited to reduce the total amount of digital video data transferred or stored, at minimal perceived quality loss. Instead of storing the exact value of every high frequency texture coefficient in the signal, compression algorithms could simply retain approximations for high frequency data with low precision, when convenient. And, at least according to their experiments, human viewers would be unlikely to perceive the difference.

Nasir Ahmed was the researcher who originally proposed using the discrete cosine transform as the mechanism for separating high frequency and low frequency texture information in 1972 (Ahmed, 1991). Since then, other mathematical frequency transforms have also been adapted to achieve a similar effect: the discrete wavelet transform (DWT) (Jensen and la Cour-Harbo, 2001), the fast Fourier transform (FFT) (Rao et al., 2010), and the discrete sine transform (DST) (Jdidia et al., 2021) are well-known examples. These other transforms have secondary benefits that make them an attractive alternative to the DCT for certain situations. But ultimately, all of these transforms can be used to achieve the same primary effect: to separate low frequency data from high frequency data so that they can be represented differently by a compression algorithm.

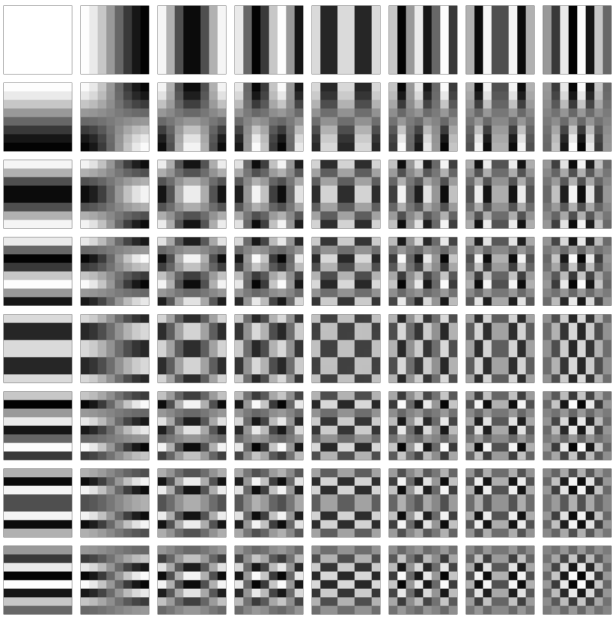


Figure 2.1: These 64 matrices represent the DCT basis functions used by JPEG to apply the 8x8 discrete cosine transform. Once a given 8x8 patch of pixels is transformed, the resulting DCT coefficients correspond to the weights of their associated basis functions depicted here.

For a square, 2-dimensional matrix $I$ with dimensions $N \times N$, the discrete cosine transform is defined by

$$\text{DCT}(x, y) = \frac{1}{\sqrt{2N}} C(x)C(y) \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} I(i, j) \cos\left(\frac{(2i+1)x\pi}{2N}\right) \cos\left(\frac{(2j+1)y\pi}{2N}\right) \tag{2.1}$$

where $C(\xi)$ is

$$C(\xi) = \begin{cases} \frac{1}{\sqrt{2}}, & \text{if } \xi = 0 \\ 1, & \text{if } \xi > 0 \end{cases} \tag{2.2}$$

This is the formulation of the DCT used by JPEG (Wallace, 1992).

### 2.2.2 Quantization

*Quantization* is the primary method by which coding systems apply lossy compression to data. Quantization works by defining a mathematical *quantizer* function and a corresponding *inverse quantizer* function, which map signal components to and from a compressed representation

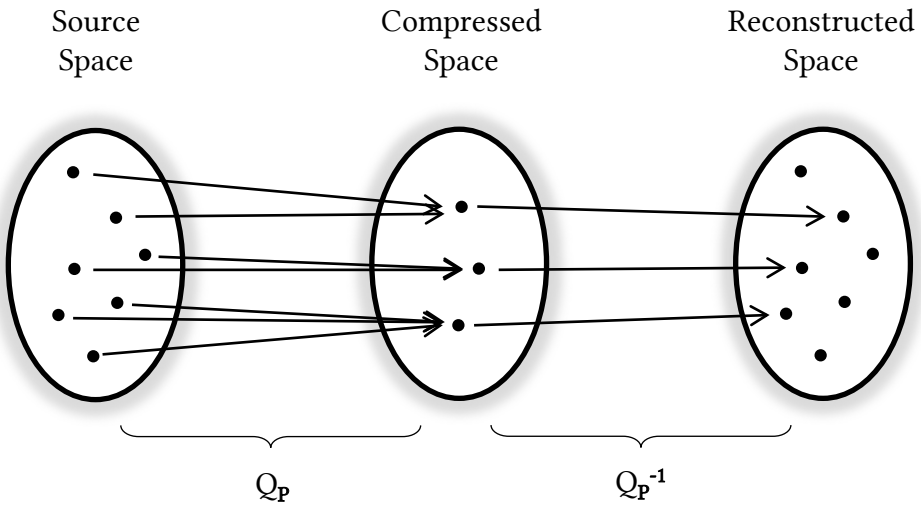



Figure 2.2: An abstract illustration of quantization is depicted. The quantizer function, $Q_{\mathbf{P}}$, maps from source space to a compressed space with lower cardinality. The inverse quantizer function, $Q_{\mathbf{P}}^{-1}$, maps from compressed space back to source space. The quantization process causes some points to be aliased during reconstruction, inducing both compression and loss.

space. By definition, the compressed space produced by a quantizer must have lower cardinality than the input space. This forces the quantizer to alias some distinct inputs to the same point in compressed space, and the inverse quantizer to choose a representative element in input space for each aliased point in compressed space. In other words, quantizing a value loses precision (see Figure 2.2).

The quantization strategy used by digital video systems is simply rounded division of bounded real input values by a positive integer constant (Ding and Liu, 1996). Mathematically, this can be expressed as follows. For input matrix $\mathbf{X}$ and constant *quantization matrix* $\mathbf{P}$, define the quantizer function $Q_{\mathbf{P}}$ by

$$Q_{\mathbf{P}}(\mathbf{X}) = \text{round}\left(\mathbf{X} \,/\, \mathbf{P}\right) \tag{2.3}$$

where $/$ represents element-wise division. The corresponding inverse quantizer function is $Q_{\mathbf{P}}^{-1}$, defined by

$$Q_{\mathbf{P}}^{-1}(\mathbf{X}) = \mathbf{X} \ * \ \mathbf{P} \tag{2.4}$$

where $*$ represents element-wise multiplication. Not only is this strategy computationally simple, but it also provides an easy way to control the amount of loss incurred for each component of the input signal. Each element of quantization matrix **P** determines the cardinality of the quantizer's range for that signal component, and therefore controls how many values in input space will alias to the same point in compressed space. By intentionally setting the values in the quantization matrix, loss can be targeted towards selected signal components.

Intuitively, decreasing the magnitude of an element in **P** results in the following behavior for its corresponding signal component:

1. Increased cardinality in compressed space
2. Fewer distinct input values aliased to the same point in compressed space
3. Increased accuracy of recovered, inverse-quantized values
4. Decreased effective compression

Increasing the magnitude of elements in **P** has the opposite effect. This inverse trade-off between accuracy of reproduction and achieved compression is a fundamental mechanism of all lossy data compression.

Quantization has wide applications throughout signal digitization and processing for its ability to easily trade off reproduction accuracy for data compression. In the field of image and video compression, which is most relevant to this thesis, it is primarily used to apply controlled loss to the coefficients produced by the DCT or equivalent frequency transform.

A typical image transform and quantization process is depicted in Figure 2.3. First, an image consisting of pixel intensity values in the range 0-255 is spatially segmented into two-dimensional, square coding blocks; 8x8, 16x16, 32x32, or 64x64 pixels are common block size choices. Each block serves as a distinct input data frame for the transform and quantization process. Next, a frequency transform like DCT is applied to each block, producing a same-sized corresponding output block of transformed, real number frequency coefficients. The transformed

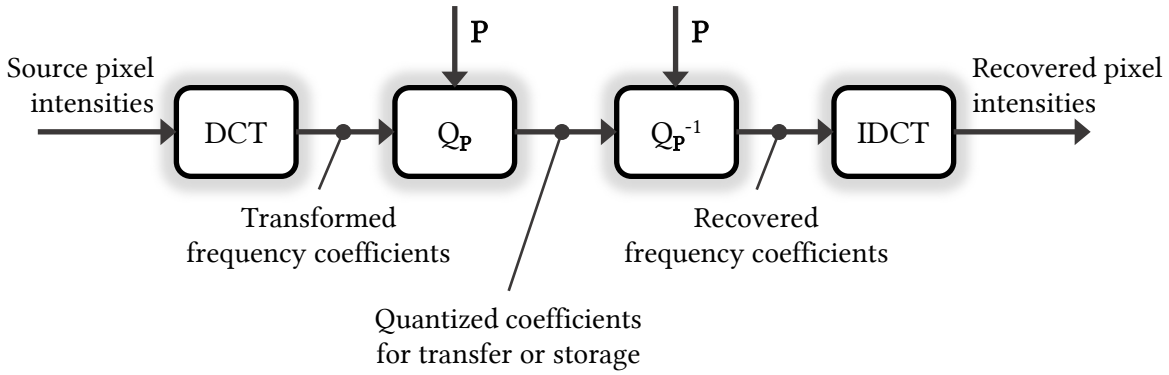



Figure 2.3: This block diagram illustrates how frequency transform and quantization are used by digital image and video systems to apply controlled loss to image signals. Input values are two-dimensional blocks of bounded pixel intensities, for example an 8x8 block of integers ranging from 0 to 255. **P** represents a quantization matrix containing a quantization parameter for each frequency coefficient in the block. The transformed and quantized signal may be stored or transferred before it is transformed back into original space.

frequency coefficients are then quantized using a predefined quantization matrix **P**, which defines how much each coefficient will be quantized. The values in the quantization matrix are set so that high frequency coefficients of less visual importance are quantized more heavily, and more important low frequency coefficients are quantized less heavily. At this point, the quantized coefficients may be transmitted or stored at reduced data rates until a reproduction is desired. At that time, the inverse quantization can be applied—using the same quantization matrix as before—followed by the inverse DCT transform. The result is a recovered approximation of the original signal, ready to be displayed or used by the application.

### 2.2.3 Color Space Transform

In addition to the discrete cosine transform, modern video coding systems also employ a color space transform to decorrelate color image data into its *luma* (i.e. relative brightness) and *chrominance* (i.e. color) components.

Humans are trichromatic organisms. The human eye has three types of "cone" receptor cells, which are sensitive to three overlapping regions of the visible light spectrum: one dominated by red wavelengths, one by green, and one by blue. This biological reality has had great influence

over the theory and design of visual systems, including image sensors, color representation, and display technology. Color itself is merely a construct used to describe how humans perceive different spectra of visible light. The most natural way of categorizing color is by describing the red, green, and blue components present in a light spectrum, just like the human eye does. This approach was first explored in the 1920s, culminating in the publication of the *CIE 1931 color spaces* and marking the first quantitative attempt at mapping colors to three-dimensional space according to human physiological perception (Ohno, 2000).

Today, image sensors and displays still model light as a combination of red, green, and blue components (RGB). This makes sense, since light is physically captured and emitted by these devices as combinations of red, green, and blue. Video compression systems, on the other hand, have no fundamental tie to an RGB representation. They benefit by transforming to a different color space where brightness has been decorrelated from the rest of the signal.

Isolating brightness information in video data has two benefits. Most importantly, the HVS is known to be more sensitive to errors in brightness when assessing quality of visual data (Winkler et al., 2001). By storing brightness separately, compression algorithms can easily target loss towards the other, less visually important aspects of the signal. But the other reason brightness is stored separately is historical. In the mid-twentieth century, older black-and-white cathode ray tube (CRT) displays were being replaced by color CRTs. To take advantage of these new displays, engineers needed a backwards-compatible media format for color video that would continue to work on older, black-and-white displays. The most straightforward solution was to multiplex a backwards-compatible, luma-only analog signal together with the color data as a side channel. This prompted researchers to develop non-RGB color spaces and corresponding transforms (Szedo, 2006).

Color space transforms can bridge the gap between the way light is sensed or displayed (i.e. as red, green, and blue triplets) and the way it is stored or transmitted (i.e. as luma and chrominance triplets). To this end, numerous non-RGB color spaces have been developed since the 1970's, including YUV, YPbPr, YCbCr, HSV, and HSL. While these spaces have varied uses

across different devices and domains, they all decorrelate brightness into its own dimension. YCbCr is used most frequently in image and video compression literature, and is standardized by the ITU-Recommendation BT.709 for use by digital HDTV applications (Pedzisz, 2013). The space is defined as a linear transform of RGB space according to the following transformations:

$$\begin{bmatrix} Y \\ C_b \\ C_r \end{bmatrix} = \begin{bmatrix} 0.2126 & 0.7152 & 0.0722 \\ -0.1146 & -0.3854 & 0.5 \\ 0.5 & -0.4542 & -0.0458 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{2.5}$$

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1.5748 \\ 1 & -0.1873 & -0.4681 \\ 1 & 1.8556 & 0 \end{bmatrix} \begin{bmatrix} Y \\ C_b \\ C_r \end{bmatrix} \tag{2.6}$$

Here, $(R, G, B), (Y, C_b, C_r) \in [0, 1]^3$, $Y$ is the luma component of the color, and $C_b$ and $C_r$ are the chrominance components. $C_b$ stands for chroma-blue and $C_r$ stands for chroma-red, representing roughly the offset from luma required to recreate the blue and red color values of the given light sample.

### 2.2.4 Chroma Subsampling

After image data is decorrelated into its luma and chrominance components, a lossy compression algorithm can apply loss to the chrominance signal with minimal loss of perception. This process is analogous to applying the DCT to decorrelate by frequency and then quantizing the resulting coefficients. However, modern video compression systems use chroma subsampling instead of quantization for applying loss to chrominance. Subsampling is the simple process of skipping samples at a regular interval. For instance, subsampling an array by a factor of 3 would mean omitting every third value in the output representation.

There are two common patterns of chroma subsampling used by video coding systems: 4:2:2 and 4:2:0. With 4:2:0 subsampling, chrominance pixels are subsampled by a factor of 2 in both the horizontal and vertical direction. This yields an overall bit rate savings of 50% when

applied to a color image. With 4:2:2 subsampling, chrominance plane is subsampled by a factor of 2 in the horizontal direction but none in the vertical dimension, yielding overall bit rate savings of 33%. As an abuse of notation, the terminology 4:4:4 is sometimes used to refer to an image that has not been subsampled at all. Thus, a 4:4:4 "subsampled" image contains the same number of pixels in the chrominance planes as it does in the luma plane.

## 2.3 Predictive Coding

Transform coding, like DCT and the color space transform, allows compression algorithms to target loss toward areas of less visual importance along the spatial dimensions. This yields huge decreases in required bandwidth with minimal reduction in perceived quality. But the bandwidth problem was not solved by transform coding alone—another contribution was also vital to the success of digital video: the development of sophisticated prediction algorithms, capable of losslessly eliminating redundancy across both spatial and temporal dimensions. This family of techniques is called *predictive coding*.

Predictive coding refers to a compression scheme in which data is segmented into a sequence of frames, symbols, or units. Each frame of data is represented by first specifying instructions on how to generate a *prediction* for the frame based on previously signaled data, and then specifying an error correction signal for correcting the predicted values. The error correction signal $R$ is called the *prediction residual*, and is equal to the difference between the prediction $P$ and the true frame data $T$.

$$T = P + R \tag{2.7}$$

If the frame being compressed is highly correlated with prior frames, then it is possible to use this prior data to form a good prediction, i.e. $P \simeq T$. In this case, the residual will be small in magnitude and uncorrelated—resembling random noise. This yields a compact representation for the frame that avoids unnecessarily re-transmitting redundant information.

Predictive coding is attractive because it reduces the task of orchestrating compression down to the more straightforward task of forming a good prediction, thereby playing to the human strength of pattern recognition. It is a perfect choice for compressing natural digital video, since adjacent frames in the temporal dimension and nearby textures in the spatial dimension exhibit so much redundancy. Modern video compression formats therefore support prediction in both temporal and spatial dimensions. Temporal prediction is sometimes referred to as *inter-frame coding* or *motion-compensated prediction*, because it involves forming a prediction using reconstructed samples from prior decoded frames. Spatial prediction is called *intra-frame coding* because it exploits redundancy in textures across a single frame.

### 2.3.1 Differential Coding: A Precursor to Motion-Compensated Prediction

Many simple compression strategies, such as *differential coding*, can be interpreted as a predictive coding technique. Consider a series of integer samples representing a monophonic, one-dimensional digital audio signal. Here, neighboring integers are highly correlated, as they approximate a continuous waveform. A naive compression algorithm for such a signal would be to simply calculate differences between adjacent samples—that is, compute and store the time derivative of the signal together with the true value of the first sample. This simple approach uses the unaltered previous sample as the prediction for the next sample. Since the same prediction algorithm is used for every sample, no "prediction instructions" need be communicated as part of the compressed representation. The prediction residual, then, simply becomes the difference between the prediction (i.e. the prior sample) and the true value (i.e. the next sample).

In fact, this exact strategy was proposed for coding analog video in 1959, as a notable precursor to motion-compensated prediction. The researchers simply computed and transmitted the difference between subsequent analog video frames as a form of compression. Naturally, this strategy does not take motion between frames into account; any motion present in the scene manifests itself as part of the residual signal. And while the development of more sophisticated motion-compensated prediction in the 1970's quickly eclipsed these early simplistic approaches,

the overall idea of removing redundancy by forming a prediction that incorporates prior data has remained the same.

### 2.3.2 Inter-Frame, Motion-Compensated Prediction

Differential coding simply reuses the previous frame as the prediction for the next frame. The major limitation of this strategy for video prediction is that it is unable to model the effect of natural motion occurring between frames. Even simple motion that is overall quite predictable, such as an object translating across the frame at constant speed, cannot be predicted by differential coding. Since motion is such a dominating (and predictable) feature of natural video, it makes sense to design prediction for it. Thus, researchers in the 1970's designed motion-compensated prediction techniques to effectively model in-scene motion.

Motion-based prediction generally works in two steps. First, a *motion estimation* or *motion detection* algorithm is applied by an encoder to analyze successive frames and estimate the motion between them using motion vectors. The resulting motion model is then used to generate instructions for how to use one frame to predict the other—this is called *motion compensation*. Suppose an encoder is generating a prediction for frame $B$, based on information previously signaled in prior frame $A$. First, the encoder partitions frame $B$ into a grid or quadtree of smaller image patches, perhaps 8x8, 16x16, 32x32, or 64x64 pixels each. Each "patch" is either called a *block*, *macroblock*, *superblock*, or *coding unit*, depending on the coding algorithm being used. Next, the encoder performs a search algorithm for each coding unit, attempting to find a nearby patch of the image in frame $A$ that is similar to the data in frame $B$. If a suitable match is found, the encoder computes the resulting two-dimensional *motion vector*, which is the pixel offset in both the x and y directions between the coding unit's spatial location in frame $B$ and the corresponding suitable patch of pixel data from frame $A$. A motion vector for a given coding unit in frame $B$ therefore identifies the corresponding location in frame $A$ where the matching prediction pixels are located. Sometimes there is no matching patch in frame $A$, in which case the search algorithm fails to find a good prediction for the coding unit. When this occurs, the encoder must fall back to a different prediction

method, such as intra-frame, gradient, or constant value prediction (discussed below). This might occur, for instance, in a frame where an object is making its first appearance in the video as it moves from an off-scene location. On the other hand, if a suitable match in $A$ is found, the encoder simply encodes the motion vector in the compressed data stream. The encoded motion vector represents the output of the motion estimation step, and conveys a model for the motion in the scene. Later, when a decoder wishes to reproduce frame $B$, it decodes the motion vector and uses it together with a decoded copy of frame $A$ to completely recreate the prediction for all coding units in frame $B$. This represents the motion compensation portion of the algorithm, since the prediction of frame $B$ compensated for the motion detected between the frames.

The characterization of motion estimation and compensation given above has been used by all major video coding standards since the 1970's. However, a number of refinements and innovations have been incorporated into newer video coding formats, allowing encoders to be more expressive and flexible when describing motion-based prediction. The result is that video data coded with modern video formats contains an expressive, meaningful, semantic prediction that not only understands the concept of motion between successive frames, but also supports rough segmentation for describing in-scene moving objects. The most important developments in prediction features are summarized below.

**Sub-pixel motion vectors**

Although early motion vectors were described using integer pixel offsets, new codecs support sub-pixel displacements in both x and y dimensions for describing a suitable prediction patch. Motion vectors that use sub-pixel precision indicate that the prediction patch should be attained by spatially interpolating the prior frame's pixel values at the specified location, typically using linear interpolation. This feature reflects the assumption that the resolution of real-world motion exhibited through time is often higher than the spatial resolution recorded by image sensors.

### Variable block sizes

Early video coding formats required encoders to divide frames into fixed-size blocks for prediction. Over time, formats have evolved to support varied prediction block sizes. Newer formats, beginning with MPEG-4 and H.264, support variable sized, non-square prediction blocks. Intuitively, objects in motion rarely project onto the imaging plane as perfectly symmetric squares, so it makes sense for prediction to evolve toward supporting arbitrary shapes. However, over-specifying the prediction block size requires extra bits in the output representation, so there is a trade-off between expressiveness and bit cost. Today, the most successful modern formats still constrain prediction regions to a limited number of predefined rectangular shapes, but are much more flexible with the sizes of these blocks than original codecs once were.

### Global motion

There are generally two types of motion present in a natural video scene: global motion and object motion. *Global motion* is the result of the video camera physically rotating or translating in space, altering the viewpoint of the scene. *Local motion* refers to motion or deformation of physical objects present in the scene. Both types of motion result in distinct changes to the two-dimensional projection of the scene being captured by the sensor as time passes.

Early efforts to generalize motion-compensated prediction included separate models for describing the global motion and the local motion present between successive frames. However, proposals for a global motion model were dropped from H.264 during the planning phase. Experiments found that local motion vectors were sufficient for modeling both local and global motion, and a separate model for global motion was not necessary. Furthermore, motion vector prediction, described below, is able to predict most global motion trends present across all coding units in a frame, with very low overhead in the output stream.

### Motion vector prediction

Motion vector prediction is the process of introducing a separate prediction model for coding the motion vectors. Spatially collocated prediction blocks often exhibit motion in the same direction—especially if a large moving object in that region spans multiple coding units. This means that nearby motion vectors are often correlated, and therefore are a good candidate for predictive coding. To exploit this redundancy, modern codecs beginning with H.264/AVC and H.265/HEVC introduce prediction models for representing the motion vectors.

Motion vector prediction works by first computing a "candidate list" of nearby, previously signaled motion vectors within the same frame to serve as the prediction for the next motion vector. The process of computing the candidate list must be completely deterministic and reproducible by the decoder. Next, the encoder simply signals which candidate motion vector to use as the prediction by encoding its index within the candidate list as part of the output stream. The decoder is expected to be able to follow along, since it presumably has decoded enough information to reproduce the complete candidate list. The winning candidate selected and signaled by the encoder should be the motion vector in the candidate list that most closely represents the true motion vector as determined by the encoder's search algorithm. Finally, after encoding the winning candidate's index, the encoder signals the motion vector "residual," which is the difference between the candidate vector's x and y values and the true x and y values for the given prediction block's motion vector.

### Bi-directional motion prediction

Motion-compensated prediction requires that a previously coded frame has been decoded and is available to be referenced when forming the prediction for the next frame. However, a richer prediction could be obtained by referencing predicted blocks from the two previously decoded frames, instead of only from the last one. This is the intuition behind bi-directional motion prediction. However, correlation due to motion in nearby frames is highest when those frames are closest in time. For a triplet of adjacent frames $F_1$ – $F_2$ – $F_3$ in order by capture time, frame $F_2$ is more highly correlated to frames $F_1$ and $F_3$, due to its adjacency to them; frames $F_1$ and $F_3$ are less correlated.

Thus, an optimal prediction is achieved when frame $F_1$ and frame $F_3$ are encoded before frame $F_2$ in the compressed bitstream, and then frame $F_2$ is encoded afterwards. This way, frame $F_2$ is able to predict in both temporal directions—backwards to $F_1$ and forwards to $F_3$. This is called *bi-directional motion prediction*.

Video codecs that support reordering frames to enable bi-directional motion prediction commonly distinguish between *display order*, referring to the order in which the frames were captured, and *coding order*, the order in which they are coded in the compressed stream. There is no fundamental requirement that coding order must match display order, so long as the decoder has enough time and memory to buffer any decoded out-of-order frames, store them to be referenced for prediction, and re-order them before they are displayed. However, dependencies between frames must be carefully tracked. Modern codecs therefore explicitly assign each frame an index indicating its display order, called its *picture order count*. This index is signaled for each frame as part of the encoded stream. Additionally, each frame is labeled based on what type of prediction it uses, determined by its coding order position within the stream. Frames that have two previously decoded neighboring frames to reference for prediction are labeled *B-frames*, indicating that they support bidirectional prediction. Frames that only have one previously decoded neighboring frame to reference are labeled *P-frames*, indicating that they support normal prediction. Finally, frames that do not have any previously decoded neighboring frames to reference are labeled *I-frames*, indicating that they support intra-frame prediction only. I-frames are a necessary part of every video, since the very first frame must be signaled without motion prediction.

### 2.3.3 Intra-Frame Prediction

Not all coding units can be predicted using motion-compensated prediction. For example, some frame regions represent new textures that were not present in previously coded frames. Sometimes, the search algorithm is unable to find a match in the nearby search region. And in the case of I-frames, a prior frame from which to predict may not even exist. In these situations, a

prediction must be formed that does not reference image content from prior frames. This is called *intra-frame prediction*.

For these coding units, video codecs provide a number of fallback strategies for forming a prediction using only pixel data from spatially adjacent, previously decoded pixels within the same frame. Usually, the only pixel values used for forming an intra-frame prediction are the reconstructed pixels in the column directly to the left of the block and those in the row directly above the block. The blocks below and to the right of the block are not generally used for prediction, since most video formats require that blocks are signaled left-to-right, top-to-bottom. Thus, the samples below and to the right are typically not yet available for prediction, since they have not yet been signaled in the output stream. Sometimes, even the values to the left and above aren't even available—such as when the block is located on the left or top edge of the frame. Thus, the algorithm for forming an intra-frame prediction must be flexible, dealing with numerous possible edge cases.

Video standards predefine a number of strategies for forming an intra-frame prediction using the available neighboring pixels. These strategies are called *intra prediction modes*. Newer codecs tend to support more intra prediction modes; for instance, H.264/AVC offers only 9 modes, while H.265/HEVC supports 35. Intra prediction modes generally may be organized into the following three categories.

**DC Prediction Mode**

When the *DC prediction mode* is selected, a single intensity value is used as the prediction for all pixels in the prediction block. The single intensity value may be encoded as part of the output stream, or, more commonly, it may be formed as the average of the available previously decoded pixels to the left and above the block. This mode is commonly used to form the prediction of the very first coding unit of the frame, since no neighboring samples are available from which to predict. It is also a good choice for predicting very "flat" regions that exhibit very little texture or intensity changes throughout the block.

#### Planar Prediction Mode

The *planar prediction mode* predicts that the block's intensity values are sampled from a plane. The plane used to form the prediction is parameterized by three noncollinear points, which are typically fitted to minimize the discontinuity between the block and any available neighboring pixels on the left and top boundaries. Since this mode uses only the available neighboring pixels to form a prediction, no additional parameters need be signaled in the output stream beyond the mode itself in order to fully recreate this prediction.

#### Directional Prediction Modes

Finally, the *directional prediction mode* indicates that the prediction should be formed by linearly extrapolating the available neighboring pixels on the left and top boundaries downward and rightward across the block, along an angle specified as a parameter in the output stream. This effectively extends the texture of the neighboring decoded block linearly across the next block, and is useful for regions where a large, homogeneous, linear texture is present across multiple coding units. Whereas H.264/AVC defines 8 discrete directions for linear extrapolation, newer formats have expanded to support more precise prediction (H.265/HEVC supports 33 discrete directional modes).

### 2.3.4 Frame Partitioning

Prediction and transform coding work best on relatively small patches of image data, because the spatial and temporal patterns they exploit are local in time and space for natural video. For instance, a single pixel's value is likely to be relatively consistent across a small number of frames; however, across a long video like a movie or live stream, that same pixel's overall intensity histogram will be more uniform. Similarly, a small region of pixels for a given frame might all have the same intensity value—even if the overall intensity histogram for the frame is uniform. By partitioning video frames into small pieces, these local spatial and temporal patterns can be described and exploited more easily.

Asserting a single optimal image patch size for a general-use compression algorithm is difficult, however. The scene's content, sensor's resolution, and frame rate all affect the size and duration of patterns present in the resulting captured intensity data. In the early days of digital video, digital cameras were relatively homogeneous, and these parameters were relatively stable for most video content. Thus, early video coding formats like H.264/AVC enforced a uniform macroblock size of 16x16 pixels, which served as the base image patch size. As sensor pixel density–and thus resolution—has increased, however, 16x16 pixels is now too small to describe or exploit textural patterns that may span across many times more pixels.

Thus, a recent movement in video format design has been to support more flexible partitioning regimes that allow for larger block sizes. H.265/HEVC was the first major standard to move away from a fixed, square macroblock. Instead, video coded with H.265/HEVC is partitioned using a quadtree, with coding units that can range from 64x64 down to 16x16. Newer formats AV1 and H.266/VVC both support 128x128-sized superblocks, which can be subdivided by the encoder into various configurations of smaller-sized rectangular blocks to best support the encoded content. This movement towards larger block sizes has resulted in huge bit rate savings, especially for high-definition and 4K content.

However, this movement also means that video coded with modern codecs contain embedded within them a much more sophisticated prediction, including spatial segmentation for each frame that identifies regions based on their coded complexity. Areas that involve high motion or detailed textures are identified in the prediction by smaller coding blocks. Areas of low motion or coarse textures are easily identifiable because they are coded using larger blocks. This dissertation claims that the sophisticated prediction of modern codecs is underutilized by modern video systems. To bridge this gap, it investigate ways in which this embedded semantic understanding of a video scene can be exposed to clients to better serve application-level goals or exploited to achieve real-time scalable coding.

## 2.4 Entropy Coding

So far, this chapter outlined the bandwidth problem facing digital video in the 1970's, and described the two major algorithmic innovations that together formed a solution: predictive coding and transform coding. However, the full picture of modern video coding is incomplete without a discussion of entropy coding. Entropy coding describes a family of lossless compression techniques concerned with assigning codewords of different lengths to the possible input symbols according to the expected probability of each symbol appearing in the data stream. The high-level intuition is that the most frequent symbols should be assigned codewords that are easier to signal, whereas rarer symbols can afford to be assigned longer codewords. Entropy coding techniques try to optimize the process of assigning codewords in a way that minimizes the output compression rate.

### 2.4.1 Shannon's Source Coding Theorem

*Information theory* is the branch of mathematics concerned with entropy coding techniques. It was founded in 1948 when Bell Labs engineer Claude Shannon published his landmark paper "A Mathematical Theory of Communication" in the Bell System Technical Journal (Shannon, 1948). In this seminal work, Shannon pointed out that the "significant aspect" of representing and signaling a message is not what is ultimately expressed, but instead the fact that the message sent was "one selected from a set of possible messages." In other words, any particular message may only be meaningfully expressed by a system that is also capable of expressing the non-selected messages.

Shannon underpinned this interpretation of communication by inventing a system of mathematics to describe his ideas. Define symbol alphabet $A$, which represents the complete set of all messages (or *symbols*) that might be conveyed. In this context, Shannon defined communication as the act of selecting and signaling a particular element from $A$. Thus, let $S$ be the random variable describing which symbol is chosen, and let $P(S = s)$ be the probability that that symbol is $s \in A$. Intuitively, Shannon argued that highly probable symbols are expected to be signaled most often, and they therefore convey less information. In contrast, low-probability symbols are more informative

to the recipient because they represent an outcome that is unexpected. To express this concept, he defined the *self-information*, or *surprisal* $I(S = s)$ of each outcome $s \in A$ to be the negative logarithm of the probability of that outcome:

$$I(S = s) = -\log_b P(S = s) \tag{2.8}$$

While the value of the logarithm base $b$ is unimportant for the following analysis, it does determine the units of self-information. Intuitively, one unit of self-information represents the amount of information gained when a single event occurs out of $b$ equally probable outcomes. When $b = 2$, self-information is expressed in units of bits. As expected, improbable symbols convey higher self-information than probable ones. Shannon further defined the *entropy* $H(S)$ of event $S$ to be the expected value of the self-information of signaling a symbol in $A$, before it is known which symbol will be signaled:

$$H(S) = \mathbb{E}[I(S)] = -\sum_{s \in A} P(S = s) \log_b P(S = s) \tag{2.9}$$

This definition of entropy implies an upper bound on the maximum compression achievable when representing stochastic information. More specifically, Shannon proved that no coding technique can represent information from a stochastic source using on average less than $H(S)$ bits per symbol in the long run. This important result is often called Shannon's Source Coding Theorem.

### 2.4.2 Arithmetic Coding

Arithmetic coding is an entropy-based technique for serializing an input symbol stream to a bitstream that supports arbitrarily precise symbol probability models and can obtain theoretically optimal compression. Due to this flexibility, it is a well-known compression technique widely used in modern video coding standards. While the details of arithmetic coding are not crucial for this dissertation, a basic understanding of its probability model management is important and is therefore outlined below.

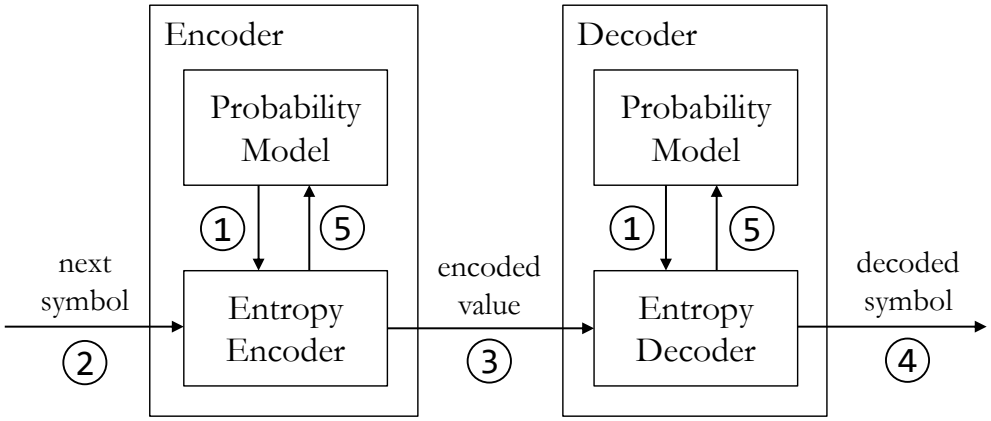



Figure 2.4: A conventional arithmetic coding structure is depicted with numbered edges expressing the data flow as a symbol is processed. ① Encoder and decoder query the probability model to determine which bit patterns to use when coding symbols. ② The encoder receives the next symbol from the input stream. ③ The entropy encoder uses the symbol's probability to encode it in the bitstream. ④ The decoder uses the symbol's probability to decode it from the bitstream. ⑤ The encoder and decoder use the symbol value to update their probability models.

Every time a symbol from the input stream is about to be processed by an arithmetic codec, a precondition is that the encoder and decoder must share a common symbol probability model. This model estimates, for each possible value of the next symbol, an associated probability representing how likely it is that that value will be the next coded symbol in the stream. The shared probability model is used by the encoder and the decoder to deterministically decide how many bits to spend when signaling each possible value, in a way that high probability values will be signaled with shorter bit patterns and low probability values will be signaled with longer bit patterns. After the true next symbol value is signaled in the bitstream, the encoder and decoder may update the state of the probability model based on the signaled value, since both the encoder and the decoder know what the value was; probability models which adjust in response to prior coded values are called *adaptive*. Furthermore, the probability model may also change for the next coded symbol based on a semantic understanding of the meaning behind that next symbol and a corresponding understanding of which symbols are likely given that meaning. Note that adapting in this way requires knowledge about the specific data being coded. Probability models that adjust based on an understanding of the symbol's contextual meaning within the larger stream are generally called *context-adaptive*.

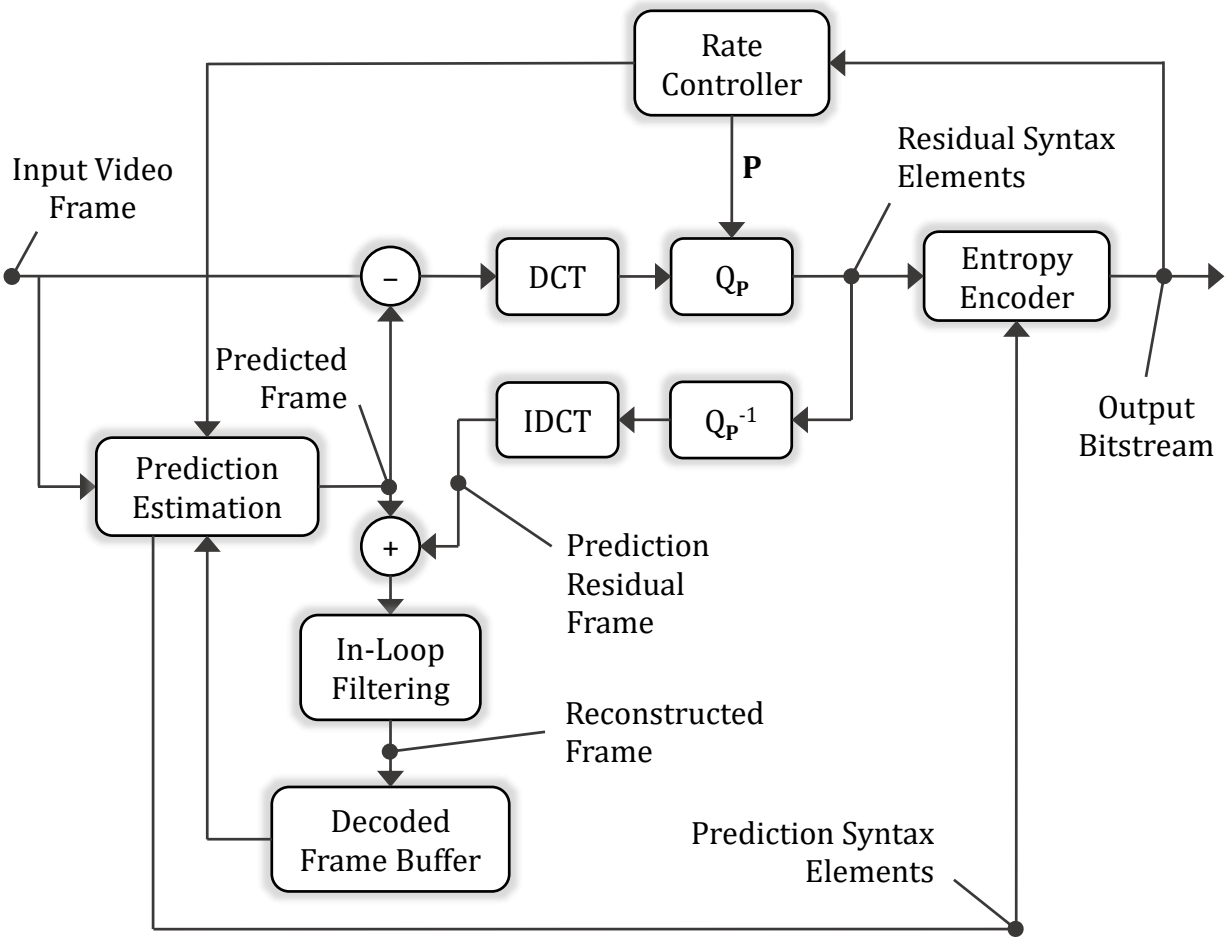



Figure 2.5: This block diagram depicts typical data flow for a hybrid prediction-transform video encoder.

Figure 2.4 depicts a high-level block diagram of a context-adaptive arithmetic codec.

## 2.5 A Complete Hybrid Coding Model

This section presents the complete hybrid coding model used by modern video formats, including both encoder and decoders. The model described herein takes a high-level approach, presenting codec elements as a block diagram. All three crucial elements described in this chapter are included by the model: predictive, transform, and entropy coding.

In the hybrid encoder depicted by Figure 2.5, each frame of raw input video is first fed into the prediction estimator, which partitions the frame into coding units. The prediction estimator performs inter-frame and intra-frame search algorithms for each coding unit to determine the best

mode for prediction. Once the optimal prediction coding decisions have been made, they are sent from the prediction estimation unit to the entropy encoder to be coded as part of the output bitstream. At the same time, they are also used to create the predicted frame. Next, the predicted frame is subtracted from the input frame to produce the prediction residual. The residual matrix is transformed by a frequency transform, like DCT, and quantized according to the quantization matrix $P$ specified by the rate controller. The result is a series of quantized coefficients, which are sent to the entropy encoder so they can be encoded in the bitstream. However, the encoder also needs to retain a reconstructed version of the frame to use for future prediction. The residual syntax elements are therefore de-quantized, inverse-frequency-transformed, and added to the predicted frame. When the reconstructed frame has been recovered, it is stored in the decoded frame buffer to be used for future prediction. Finally, note that many formats require that in-loop filtering is applied to all reconstructed frames to reduce blocking artifacts that appear in the reconstructed signal when quantization is applied separately across adjacent transform blocks.

The corresponding hybrid decoder is depicted in Figure 2.6. An entropy decoder decodes the input bitstream, converting the encoded codewords into symbols that can be interpreted as syntax elements. To decode a frame, the system first splits the decoded symbols into prediction elements and prediction residual elements. The prediction elements inform the decoder how to create a prediction for the next frame, by using previously decoded frames from the decoded frame buffer. The prediction residual is also recovered by de-quantizing and inverse-frequency-transforming the prediction residual elements from the input bitstream. Once both the prediction and the prediction residual have been recovered, they are added together to form the reconstructed frame. Finally, in-loop filtering is applied to all reconstructed frames, if required by the video coding format being used. The filtered, reconstructed frames are buffered in the decoded frame buffer so they can help form the prediction of future frames. They also form the output of the decoding process.

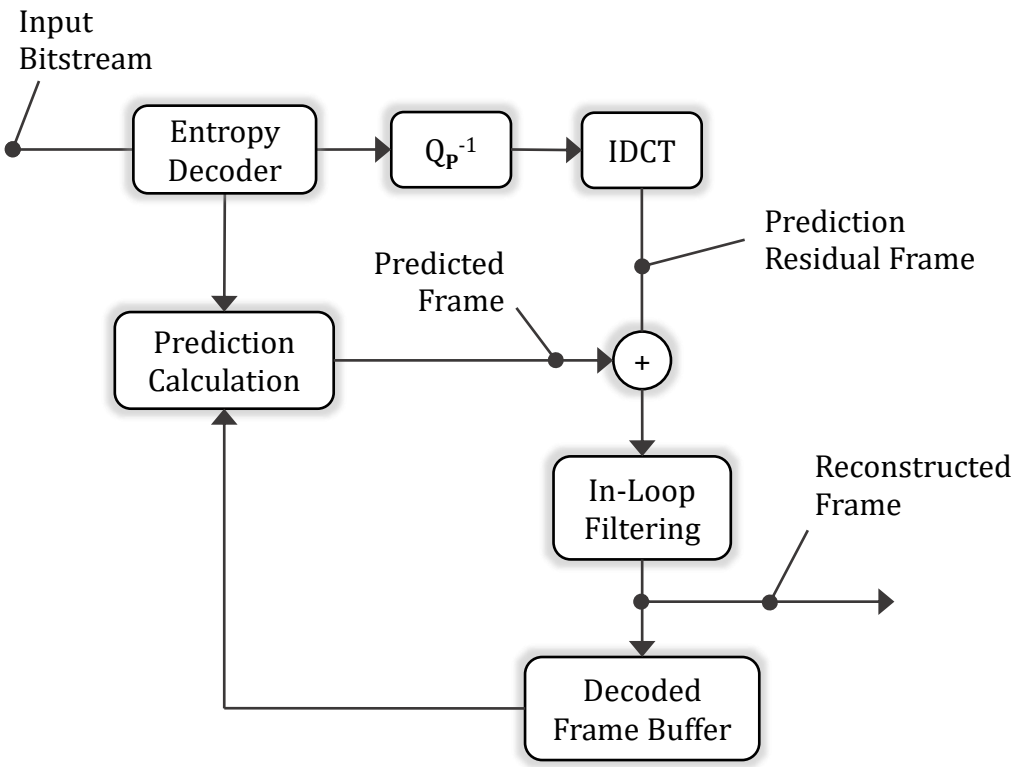



Figure 2.6: This block diagram depicts typical data flow for a hybrid prediction-transform video decoder.

## 2.6 Digital Video Coding Formats

In the decades since the digital video revolution, video coding standards have iterated on the basic hybrid model described above. New format releases aim to support new innovations in video compression, new display and sensor capabilities, new video use cases, and new types of clients. Overall, the trend has been towards more expressive predictions, higher computational costs for encoding and decoding, and higher compression ratios, especially for high-resolution content. Table 2.2 lists various descriptive parameters for recent video format releases, illustrating the community's race to design codecs that support content for larger screens at lower bandwidths.

| | Release Date | Max Block Size | Partitioning | Transform | Publisher |
|---|---|---|---|---|---|
| **MPEG-2** | 1994 | 8x8 | Macroblock | DCT | MPEG |
| **MPEG-4** | 1999 | 16x16 | Macroblock | DCT | MPEG |
| **AVC/H.264** | 2003 | 16x16 | Macroblock | DCT | JVT |
| **VP9** | 2012 | 64x64 | Tree | DCT/DST | Google |
| **HEVC/H.265** | 2013 | 64x64 | Quadtree | DCT/DST | JVT |
| **AV1** | 2018 | 128x128 | Tree | DCT/ADST | AOMedia |
| **VVC/H.266** | 2020 | 128x128 | Multi-type Tree | DCT/DCT7/DCT8 | JVT |

Table 2.2: This table highlights differences in representation between the major video coding standards released between 1994 and 2020.

## 2.7 Assessing Coding Performance

Finally, this section introduces the necessary theory used by the rest of this dissertation to benchmark video compression algorithms, compare their effectiveness, and assess their relative strengths. Traditionally, competing video compression algorithms and coding systems are compared based on (at least) the following three qualities:

1. **Reproduction Quality** – The mathematical or perceived quality of the reconstructed video at the decoder. This is particularly important for lossy compression, which results in an imperfect reproduction. Various algorithms for assessing reproduction quality have been proposed. The most common in the literature are PSNR and SSIM, defined below.

2. **Compression Rate** – The rate of bits required per second of video for the compressed representation. Video encoded at higher bit rates demand more network transfer and storage resources, while those at lower bit rates require less resources.

3. **Codec Complexity** – The computational complexity required to encode and decode the video. This becomes an important consideration when designing codecs for real-time applications as well as systems that need to support scaling the number of users.

Other trade-offs exist between compression algorithms, too, such as latency, scalability, receiver flexibility, support for new use cases, etc. However, the three qualities above are most commonly presented in the literature, reflecting the community's historic interest in algorithms that optimize specifically for these metrics. This dissertation challenges that assumption, arguing that as storage, network, capture, and display technology continues to rapidly advance, receiver flexibility has usurped reproduction quality, compression rate, and codec complexity as the dominant compression goal.

### 2.7.1 Peak Signal-to-Noise Ratio (PSNR)

*Peak Signal-to-Noise Ratio* (PSNR) is a metric imported for lossy image and video compression, originally taken from the signal processing community for quantifying the relative amount of noise present in a signal. PSNR is the ratio between a signal's maximum possible power and the average power of the noise present. PSNR is often presented in decibel units.

Suppose a still, monochrome image $I$ with dimensions $m \times n$ is encoded using lossy compression such that a corresponding decoder reproduces image $R$. The *mean squared error* (MSE) between $I$ and $R$ can be expressed as

$$\text{MSE}(I, R) = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} (I_{i,j} - R_{i,j})^2 \tag{2.10}$$

Here, $I_{i,j}$ represents the intensity of the pixel in row $i$, column $j$ for image $I$. $R_{i,j}$ represents the intensity of that same pixel in image $R$. MSE represents a simplistic metric for comparing images $I$ and $R$, but it is absolute with respect to the size of the signal—that is, the scale of MSE values are dependent on the the dynamic range of the image intensities. PSNR fixes this problem by expressing a ratio.

The PSNR of reconstructed image $R$ can be computed, relative to the original image $I$, as the logarithm of the ratio between the square of the maximum possible intensity value $i_{\text{max}}$ and the mean squared error between $I$ and $R$. For images with pixel bit depth $8$, this means $i_{\text{max}} = 2^8 - 1 = 255$.

$$\text{PSNR}(I, R) = 10 \log_{10} \left( \frac{i_{\text{max}}^2}{\text{MSE}(I, R)} \right) \tag{2.11}$$

PSNR can easily be extended to apply to video data by computing the PSNR between each pair of corresponding frames in the image. The average PSNR across all frames is often reported for a segment of video. Typical PSNR values for images and video coded using lossy compression are between 25 and 50 dB. Higher values represent less noise, and therefore a more faithful reproduction. Thus, a PSNR value can be thought of as a proxy for reproduction quality.

### 2.7.2 Structural Similarity Index (SSIM)

PSNR has received criticism for its inability to assess structural reproduction and other qualities in an image that may be more important to a human viewer. For example, consider two images, $I$ and $I'$ such that $I' = I - 5$ (assuming a dynamic range of 0–255). The PSNR between $I$ and $I'$ would be fairly low, indicating that $I'$ is a low quality reproduction of $I$—even though a human may not be able to perceive the difference on a typical display.

Thus, an avenue of research has been devoted to developing so-called *perceived quality* metrics as an alternative to PSNR. Perceived quality metrics are engineered to respond to certain image qualities that may be more noticeable to human viewers. As a consequence, these metrics are typically much more complicated than PSNR.

*Structural Similarity Index* (SSIM) is one such perceived quality metric. SSIM is the sum of three comparison metrics: luminance ($l$), contrast ($c$), and structure ($s$). Each of these sub-metrics intends to assess a separate quality to which humans have been shown to be sensitive.

$$l(I,R) = \frac{2\mu_I\mu_R + c_1 i_{\text{max}}}{\mu_I^2 + \mu_R^2 + c_1 i_{\text{max}}};\ c_1 = 0.01 \tag{2.12}$$

$$c(I,R) = \frac{2\sigma_I\sigma_R + c_2 i_{\text{max}}}{\sigma_I^2 + \sigma_R^2 + c_2 i_{\text{max}}};\ c_2 = 0.03 \tag{2.13}$$

$$s(I,R) = \frac{\sigma_{IR} + c_3 i_{\text{max}}}{\sigma_I\sigma_R + c_3 i_{\text{max}}}\ c_3 = 0.5c_2 = 0.015 \tag{2.14}$$

Here, $\mu_I$ and $\mu_R$ represent the mean pixel value of images $I$ and $R$, respectively; $\sigma_I$ and $\sigma_R$ represent the variance of pixel values in the images, and $\sigma_{IR}$ represents the covariance of pixel values between the images. Like before, $i_{\text{max}}$ represents the maximum possible intensity value, which is $2^8 - 1 = 255$ for 8-bit images.

Once $l$, $c$, and $s$ have been computed for a noisy image (or image patch), the SSIM can be computed as the sum of the three components:

$$\text{SSIM}(I, R) = l(I, R) + c(I, R) + s(I, R) \tag{2.15}$$

### 2.7.3 Rate-Distortion Theory

In "A Mathematical Theory of Communication," Claude Shannon introduced *rate-distortion theory* as an important branch of information theory (Shannon, 1948). This branch of mathematics is concerned with optimally transmitting information across a noisy channel in a way that minimizes bit rate and simultaneously prevents the resulting noise from exceeding a predefined distortion level. The inverse relationship between bit rate and distortion explored by rate-distortion theory is fundamental to all lossy compression theory, including that of video coding algorithms.

Lossy compression algorithms offer a fundamental trade-off between bit rate and distortion, which is described in Shannon's seminal work: lower bit rates can be achieved only at the cost of increased distortion (i.e. error) in the reproduction. Conversely, if distortion cannot be tolerated, a more accurate reproduction can only be guaranteed at the cost of a high bit rate encoding.

In the literature, it is typical to illustrate the trade-off between bit rate and distortion achieved by a given video compression algorithm as a curve on a *rate-distortion graph* (R-D graph). By convention, R-D graphs show target bit rate on the x-axis and an achieved quality metric on the y-axis. Generating a R-D curve for a particular video content source and a given video codec requires encoding the same content multiple times at different target bit rates, and plotting the achieved reproduction quality for each target rate. The result is a curve that generally arcs from the high quality, high bit rate quadrant of the graph down to the low quality, low bit rate quadrant. The shape of this graph indicates how the codec performs at different bit rates for the given content. A "good" codec is able to maintain a relatively high quality despite decreases in bit rate, and depicts a smooth R-D curve to indicate apt control over the trade-off between bit rate and quality.

# CHAPTER 3

## Existing Adaptation Strategies

Chapter 2 described the hybrid prediction-transform coding techniques used by modern video coding standards to compress digital video content. For the most part, these techniques have successfully mitigated the bandwidth challenge, allowing digital video to be encoded at a reasonable bandwidth. Nonetheless, the past few decades have seen significant advances in capture, display, and application technology for digital video. Today, it is no longer safe to make the same assumptions about a client's available bandwidth, quality, or computational capabilities when a request for digital video content is made.

This chapter addresses the *heterogeneous client problem* facing the online video industry today: as use cases diversify for video, existing codecs and server infrastructure is being strained. No longer is a single encoding of a given video sufficient for all use cases; instead, video must be *adapted* dynamically to fit clients' needs. Next, existing strategies for video adaptation are presented, organized in three categories: scalable codecs, adaptive streaming, and on-demand transcoding. These approaches represent three different system architectures for providing tailored versions of a video in response to diverse client requests.

All major video distribution organizations today use one or a combination of the techniques described in this chapter to adapt source video and other multimedia content to diverse clients. For each approach, this chapter identifies its strengths, limitations, assumptions, and selected representative works.

## 3.1 The Heterogeneous Client Problem

As explained in Chapter 1, recent editions of the Cisco Visual Networking Index have identified and quantified the following three trends in consumer demand for online video consumption.

1. **Diverse Content** – Video content is diversifying as a result of improvements in the capabilities of both sensor and display technology. New types of content include high dynamic range (HDR), 4k, 360° video, virtual reality, and live streaming.

2. **Diverse Receivers** – As handheld technology has improved, video receivers have become more diverse, capable, and powerful. Examples include multimedia streaming dongles, smartphones, tablets, laptops, cars, internet of things (IoT) devices, machine learning endpoints, and desktops.

3. **Diverse Applications** – To take advantage of improved display, sensor, and receiver capabilities, video-enabled applications are also advancing. New use cases for video include machine-to-machine, machine learning, autonomous driving, body cameras, live streaming, and mobile video.

These trends have shown no signs of slowing down, and may even have accelerated during the COVID-19 pandemic. A key assumption of this dissertation is that they will continue through the 2020s.

The recent developments in video systems outlined above are exciting because they represent ways in which technology can better interface with and improve the human experience. However, they also represent important challenges for existing infrastructure, and raise questions about scalability and equity of access for large audiences. As receivers, content, and applications continue to diversify, these challenges are amplified.

Slowly, tension is becoming apparent. On one hand, video receivers, content, and applications are quickly diversifying. On the other, existing, aging video codecs—which were not designed to handle diverse requests on par with modern demand—are struggling to meet these needs. Major

industry players like Netflix and Google so far have been able to keep up by investing millions of dollars in server infrastructure. But as the trends of client diversification and ever-increasing demand continue, it is uncertain how long this approach will be feasible. There is also concern that smaller video distributors which do not have the same level of financial security as the top tech companies will struggle to provide a high quality of service for diverse clients.

This dissertation labels the tension between receiver diversification and existing compression technology *the heterogeneous client problem*. Ultimately, we claim that this tension will recalibrate the focus of new video coding formats so that they prioritize flexibility and receiver-driven use cases first. Evidence pointing towards this shift already exists in the literature and in industry video system trends. Recent publications and calls-to-action have focused on secondary (but growing) use cases for video, including virtual reality applications, systems where machines are the receiving endpoint of video content, 3D rendered content, and requests for video based on regions of interest. At the same time, video content distributors are already pouring millions of dollars into infrastructure to create systems that are capable of providing video to heterogeneous clients across the globe in real time.

## 3.2 Taxonomy

Solving the heterogeneous client problem requires an *adaptive* system that can respond to diverse client requests and provide different representations of the source content to each. The remainder of this chapter is dedicated to categorizing and describing the existing solutions to this problem. Prior work in this area can generally be categorized into the following three categories of approach:

1. **Scalable codecs** solve the heterogeneous client problem at encoding time, before the video content has been requested by a client.

2. **Adaptive streaming** techniques, also called *simulcast*, solve the heterogeneous client problem at the application layer by treating the video's encoding as a black box.

3. **On-demand transcoding** techniques solve the heterogeneous client problem at request time by integrating the process of transcoding the video with the act of serving it to clients.

## 3.3 Scalable Codecs

*Scalable codecs* are a family of video coding techniques that aim to address the heterogeneous client problem during the video encoding process. The idea behind a scalable codec is to encode the content to an output stream in a way where receivers can stop decoding in the midst of the stream, and still produce an approximation of the content (Amon et al., 2008). Receiving and decoding a higher percentage of the stream results in a closer approximation of the content, whereas truncating the stream earlier yields a lower-quality reproduction. Only if the full stream is decoded can the complete, full-quality version be obtained. Content encoded with a scalable codec allows the decoder to "scale" the quality of the reproduction by deciding how much of the stream to receive and decode.

The primary benefit of a scalable approach is to give decoders the ability to easily trade off between quality of reproduction and bandwidth usage without requiring server intervention. A server can provide the exact same scalable video encoding to all clients, but each client can make a unique, tailored decision about how much of the stream to receive and decode according to its available resources and application needs.

However, designing an effective scalable codec is difficult. Scalable encoders must simultaneously optimize two often opposing goals: on one hand, the traditional data compression goal of eliminating redundancy in the output stream; on the other, ensuring that the output stream has the desired scalable substructure where heterogeneous clients can stop at different points in the stream and recover a usable approximation. In the following sections, a few approaches for designing scalable codecs are described including layered coding, bit plane coding, and wavelet transform coding.

Scalable coding for video was an active area of research in the early 2000's, around the time that AVC/H.264 was released. Most video encoding formats, including AVC/H.264 and

HEVC/H.265, and AV1, are accompanied by officially approved scalable extensions. However, scalable codecs are rarely used in practice, as adaptive streaming provides a much more straightforward system design that greatly simplifies the process of deploying video web servers to handle diverse requests.

### 3.3.1 Scalable Dimensions

Scalable codecs allow receivers to easily trade off between quality and bit rate, but the client has very limited control over which features of the video are omitted for lower bit rates. That decision is made at encoding time, when the encoder selects which dimension or dimensions of the signal will be scaled. Scalable codecs support a number of predefined *scalable dimensions* from which the encoder can choose for scaling the bit rate. The following dimensions have been proposed and implemented for scaling video data using various coding formats.

- **Signal-to-Noise Ratio (SNR) Scaling** – As more data is decoded, the reproduced pixel intensities move closer to their true values.
- **Frame Rate Scaling** – As more data is decoded, the reproduced video's frame rate increases.
- **Resolution Scaling** – As more data is decoded, the spatial resolution of the reproduced video increases.
- **Dynamic Range Scaling** – As more data is decoded, the dynamic range of the recovered pixel intensities is increased.
- **Color Gamut Scaling** – As more data is decoded, the color gamut of the recovered frames widen.

### 3.3.2 Layered Coding

*Layered coding* is an organizational strategy used by many scalable codecs in which the encoded content is partitioned into multiple streams, called *layers*. A decoder that decodes all layers

can recreate the full quality representation of the content. However, decoders may instead choose to decode only some of the layers, producing a lower quality approximation at a lower bit rate. This allows decoders to scale quality and bit rate. The layers generally follow a predetermined dependency hierarchy, meaning that not every layer is useful on its own; if a layer has dependencies, those layers must be decoded first. One layer, called the *base layer*, can be independently decoded to provide the lowest quality version of the content—this is the starting point for all decoders. The rest of the layers are called *enhancement layers*, and are used to improve the quality of the base layer's reproduction by incorporating additional information. All decoders must start with the base layer, and then follow the dependency graph to decode and incorporate enhancement layers until the desired reproduction quality is attained.

**Multiple Description Coding (MDC)**

Layered coding is related to but distinct from a similar coding technique called *multiple description coding* (MDC). With multiple description coding, an encoder produces multiple distinct, independent, low-quality versions of the source content—each of which is called a *description*. Any single description can be decoded independently to reproduce a low-quality reconstruction of the stream. However, by receiving and decoding more than one description in tandem, a decoder can combine them in some way to recover a higher quality version of the content. The major difference between MDC and layered coding is the base/enhancement layer dependency hierarchy that allows layered encoders to assume the presence of other layers at the decoder. With MDC, there is no assumed hierarchy between descriptions. For video coding and many other applications, this means that any two distinct descriptions would contain significant duplicated information associated with ensuring that a minimal quality reproduction of the content is obtained from decoding the description by itself.

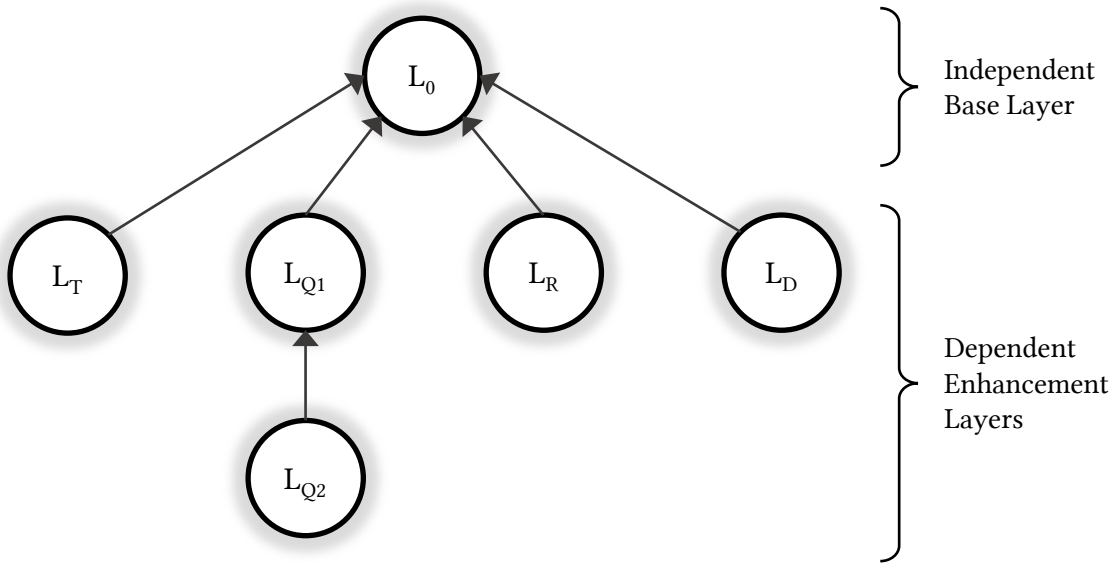



Figure 3.1: This directed graph illustrates how a scalable codec might produce layers for a video. Nodes represent layers; edges indicate dependencies between layers. $L_0$ represents the base layer, a non-scalable representation of the content with no dependencies. $L_T$, $L_{Q1}$, $L_{Q2}$, $L_R$, and $L_D$ are dependent enhancement layers. $L_T$ provides the information necessary to double the frame rate; $L_{Q1}$ and $L_{Q2}$ successively improve the SNR quality; $L_R$ doubles the resolution in both dimensions; and $L_D$ increases the dynamic range.

**Advantages**

Layered coding is both intuitive and modular. Layered coding provides an intuitive way of organizing content as part of a scalable codec, because the layers form logical units where each provides a specific type of enhancement to the base stream. For example, as depicted in Figure 3.1, one layer might provide the information necessary to double the frame rate; another might scale the resolution by a factor of two; a third layer might increase the dynamic range of the reproduced video. Two additional dependent layers might improve the SNR quality of the video in two successive steps. Decoders can select which enhancement layer(s) best match their display capabilities, available bandwidth, and application needs.

By enforcing dependencies between layers, layered coding reduces redundancy in the output stream, yielding higher compression rates than if no dependency graph was enforced. For example, consider a series of two or more dependent enhancement layers which provide improved

SNR quality. Since the layers are dependent, the representation of each layer can rely on the fact that the prior layer has already been decoded and integrated into the reproduction. Thus, successive layers can focus on improving the already enhanced content, without re-coding the same corrections. The result is a single, compact encoding that supports modular, scalable decoding with minimal redundancy.

Finally, the structure of layered coding naturally supports backwards-compatibility with non-scalable decoders. To achieve this, the independent base layer can be formulated as a standard-compliant, non-scalable, low quality coded version of the content. Off-the-shelf, non-scalable decoders would therefore be able to decode and display the base layer, but would be unable to incorporate information from the enhancement layers. Only full scalable decoders would be able to decode the full-quality content.

**Disadvantages**

The benefits of layered coding come at the cost of additional computational complexity at both the encoder and decoder, when compared with an equivalent non-scalable alternative. Video data and stream dependencies must be managed carefully by the encoder for all layers concurrently, resulting in increases in memory requirements and computational complexity. Similarly, decoders interested in reproducing high-quality versions of the content must decode and incorporate all available layers. Decoding a high number of layers generally corresponds to higher computational requirements. The layered decoding process is inherently sequential and not easily parallelizable, due to the dependencies between layers.

Moreover, there is compression overhead associated with providing a layered version of a video when compared to a non-scalable version. This is because layered approaches are inherently suboptimal when paired with predictive coding techniques, which are used heavily in video coding (Rose and Regunathan, 2001). Predictive coding involves using prior decoded information to form a prediction for future frames. However, with a layered approach, the availability of prior data at the decoder is thrown into question. In particular, the layer dependency graph must be respected by

the prediction process. No prediction can be made while coding a layer that extrapolates data from another layer, unless the other layer is a dependency. The power of prediction is therefore weakened by layered coding, especially for layers with few dependencies. This phenomenon is most obvious with the base layer. Since the base layer has no dependencies at all, any predictions made during the base layer's coding process can only extrapolate from prior data in that layer. This means that the low quality version of prior frames must be used when predicting for the base layer, even if the decoder has higher quality prior frames available from decoding the enhancement layers.

### 3.3.3 Other Scalable Approaches

Overall, layered coding has seen the most success in scalable video codec design, likely due to its intuitive modular design. Despite this success, a number of other scalable technique have also been proposed and demonstrated in the literature. Below, these approaches are briefly described.

#### Bit Plane Coding

One area of research proposed by researchers in the scalable video coding community during the early 2000's is *bit plane coding* (Mayer et al., 2002). Bit plane coding relies on the knowledge that a byte's most significant bits are more important to the magnitude of the value than the lower order bits. The idea is to prioritize higher order bits when transmitting bytes to a bit stream. Consider a two dimensional integer matrix where each element is represented by a single byte. Such a matrix may represent quantized frequency coefficients or raw pixel values for an image or frame. The idea is to separate the bits that form the element bytes into eight separate *bit planes*; that is, one plane to store the highest order bits from each element, one plane to store the second-highest order bits from each element, and so on down to the least significant bits. The resulting planes are then written to the bit stream in order from most significant to least significant.

By encoding a matrix using bit plane coding, the resulting stream exhibits the quality that the precision of the stored values increases as more of the stream is decoded. Decoders that do not

need full precision of the matrix elements are free to abandon decoding midway through the stream, as soon as sufficient precision has been received. A matrix encoded using bit plane encoding is therefore scalable with respect to the precision of its values.

Bit plane coding has been used for scalable coding in multiple commercial image compression standards, including the scalability extension of JPEG and JPEG-2000 (Wallace, 1992; Marcellin et al., 2000). It has also been explored as a technique for coding video content (Radha et al., 2001; Wu et al., 2000; Wang et al., 2002).

#### Pyramid Coding

The *pyramid representation* is a well-known approach for supporting spatial scalability for images (Burt and Adelson, 1987; Tan and Ghanbari, 1995). With this strategy, a low-resolution version of an image is first encoded at the beginning of the stream. Next, a series of residual layers are encoded that provide the necessary information to expand the resolution of the image. Decoders can choose how many layers to incorporate according to their desired final resolution.

Various mathematical operations have been proposed for decomposing and recovering an image according to a pyramidal structure. For instance, Gaussian or Laplacian filters may be applied to the image before it is down-scaled or up-scaled (Burt and Adelson, 1987). Another approach is to decompose the image using a Haar wavelet transform, such that each residual enhancement layer adds new higher frequency data into the up-scaled reproduction (Kekre et al., 2010).

### 3.3.4 Scalable Extensions to Widely-Used Formats

Many existing video coding standards, including AVC/H.264 and HEVC/H.265, are accompanied by officially supported scalable extensions that dictate how the formats can be altered to support scalable requests. Scalable Video Coding (SVC) is the published extension for AVC/H.264, and Scalable High Efficiency Video Coding (SHVC) is the analogous scalable extension for HEVC/H.265. As discussed below, these have not yet seen notable traction in practice, due in part

to the added computational and logistical complexity that accompanies supporting the extensions at both the encoder and the decoder.

Both HEVC and AVC are accompanied by officially approved scalable extension standards, termed Scalable HEVC (SHVC) and Scalable Video Coding (SVC), respectively (Boyce et al., 2016; Schwarz et al., 2007). Leading up to the release of these extensions, a number of proposed techniques for scalability were considered for inclusion in the final publication. (Ohm, 2005) lists some of these approaches. In practice, sing SHVC and/or AVC requires a server-side transcoding step, as the original stream must be transcoded to a scalable format after capture time but before request time.

### 3.3.5 Limited Support for Scalable Decoding

A major practical obstacle that has hampered real-world success of layered coding and other scalable approaches—including the officially released scalable extensions listed above—has been limited support by decoders and receiver devices. In theory, a scalable encoding of video content can provide optimal experiences to a wide range of clients. However, this only works if the clients are capable of decoding the scalable codec. In practice, scalable decoders are not widely available. Thus, many real-world clients are unable to take advantage of the enhancement layers and would be limited to receiving and decoding only the low-quality, non-scalable base layer. Due to limited decoder support for scalable codecs, video providers often choose alternate approaches like adaptive streaming to provide non-scalable standard-compliant encodings to diverse clients.

## 3.4 Simulcast / Adaptive Streaming

Scalable video coding works by reorganizing video data during the encoding process so that subsets of the encoded data stream can be decoded independently to produce a lower-quality approximation of the content. In contrast, *adaptive streaming* works entirely outside the encoding process, in the application layer. The source video content is completely decoded to pixel space and then re-encoded multiple times to create a set of fully independent versions of the content. Diverse

clients can request the version of the content which best suits their use case, and it is statically *streamed* to them. Adaptive streaming is characterized by the following traits:

1. **Application layer adaptation** – Video coding is treated as a black box. An off-the-shelf, non-scalable codec is used, and the only encoder configuration applied is the value of the coding parameter(s) that should be adapted.

2. **Separate encoder and content server** – The encoding process is completely independent from the act of serving the content. There is no feedback loop between the content server and the encoder. The encoder must decide in advance which parameters and which values will be scaled, without any real-time input from client or server.

3. **Independently coded streams** – Unlike scalable codecs, no attempt is made to eliminate redundancy in the transcoded representations. Each transcoded representation is encoded as a completely independent stream, even though this results in duplicated information. Adaptive streaming is therefore sometimes called *simulcast*, because multiple independent versions of the same content are simultaneously broadcast to clients.

Figure 3.2 depicts adaptive streaming as a block diagram. The source device captures and encodes the video content, producing an initial representation. Next, it uploads the encoded data to a transcoding server which transcodes the video multiple times, producing multiple independent output streams with different coding parameters. These transcoded versions of the video are then stored on the web server, and clients are able to request any of the pre-coded versions. No coding dependencies exist between versions of the content.

Adaptive streaming has been widely adopted and is often used in practice by content providers to support diverse requests for multimedia data (De Praeter et al., 2017). Numerous standards for adaptive streaming have been published by different vendors, summarized in Table 3.1 (Müller et al., 2012; Fecheyr-Lippens, 2010; Bocharov, 2009; Sodagar, 2011).

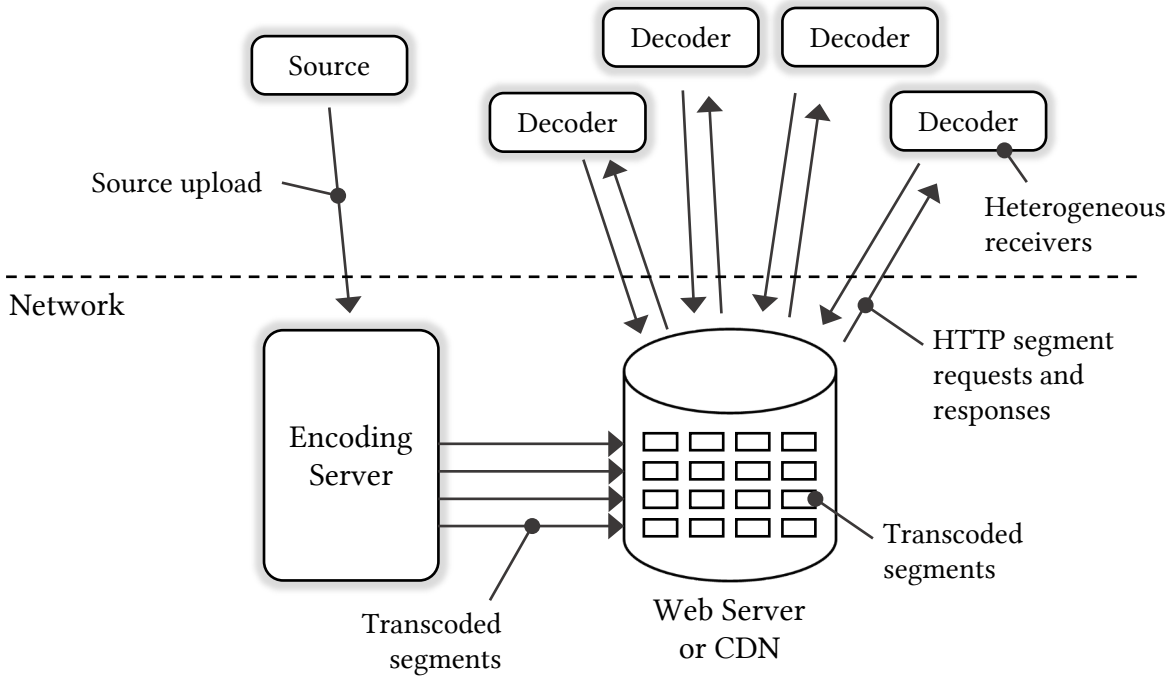



Figure 3.2: An adaptive streaming system diagram is depicted. Video content is uploaded from a source device across a network to an encoding server, which transcodes the content multiple times to produce a set of independently coded segments. These segments are stored on a web server or CDN and served to clients via HTTP.. Heterogeneous clients asynchronously request different segments of the video, according to their available network bandwidth or application needs.

### 3.4.1 Adaptive Bitrate Streaming

When an adaptive streaming system is deployed specifically to allow decoders to scale the bit rate of the received content, it is called *adaptive bitrate streaming* (ABR streaming). For video content streamed over the internet, variable transfer bit rates are common—bandwidth varies from client to client, and sometimes even the throughput of a single connection fluctuates. ABR streaming can mitigate these issues by allowing clients to adapt to changes in their available network bandwidth. Thus, ABR streaming has quickly become a particularly common application for adaptive streaming technology.

| Name | Acronym | Vendor |
|---|---|---|
| Dynamic Adaptive Streaming over HTTP | DASH | MPEG |
| HTTP Dynamic Streaming | HDS | Adobe |
| HTTP Live Streaming | HLS | Apple |
| Smooth Streaming | | Microsoft |

Table 3.1: This table highlights differences in representation between the major adaptive streaming standards in use today.

### 3.4.2 Dynamic Adaptive Streaming over HTTP (DASH)

Dynamic Adaptive Streaming over HTTP (DASH) is an implementation of adaptive streaming that is codec-agnostic and exclusively uses HTTP for network transfer (Sodagar, 2011). DASH works by temporally partitioning multimedia content into small *segments*, each containing about 5-15 seconds of non-overlapping content. Each segment is then independently transcoded multiple times using different encoding parameter values. Ultimately, this process yields a large number of short, independent video fragments, which are stored on the file system of a HTTP web server. Clients can select and request these fragments, using them to piece together the full content on the client side.

By transcoding each segment multiple times and varying the encoding parameter values, a range of options are produced for clients to choose from when requesting that segment of content. To reproduce the full content, clients must select and request a transcoded version for each temporal segment. Clients choose which transcoded version of each segment to request based on available client-side resources or application-level goals. Once the requested segments are received, the client plays them back sequentially to recreate the full content. Thus, the system is scalable because it allows clients to "scale" one or more encoding parameters by selecting which pre-transcoded version to download when requesting each segment of content. Furthermore, it allows clients to adjust their decisions throughout the content. This is important, for example, if the client is attempting to respond to system changes outside their control (e.g. network fluctuations).

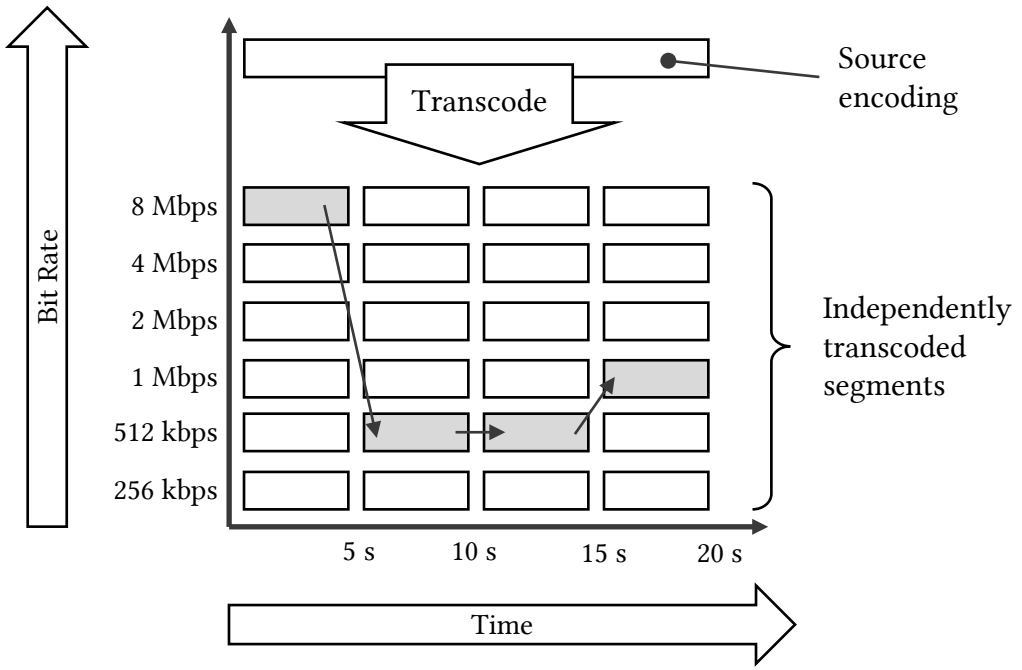



Figure 3.3: This diagram shows how a source encoding may be transcoded to produce smaller, temporally synchronized video segments at various bit rates for use in ABR streaming. Each rectangle represents an independently coded video segment, ordered by playback time on the x-axis and transcoded bit rate on the y-axis. The source encoding was originally recorded at a higher bit rate, but was transcoded down to produce the lower bit rate segments. For each time period, clients choose which segment to download according to their available network bandwidth. A sample client download path is depicted by the shaded segments and arrows. This client may have detected a decrease in network bandwidth after downloading the first video segment, and adapted by requesting a lower bit rate for the second and third segments.

When a client requests multimedia content using DASH, it first requests a special manifest file from the server, called a *Media Presentation Description* (MPD) file. The MPD manifest for a particular video describes how the video is segmented, and lists the available transcoded versions for each segment. The client uses the information in the manifest file to request the segment encodings that are most appropriate. Since this process takes place over HTTP, this results in a flurry of independent HTTP requests, one for each segment encoding that the client requests. The server simply responds with the requested pre-computed fragments stored on its file system. This minimizes server-side per-request computation, while still providing each client with an encoding that approximately meets its needs.

Figure 3.3 depicts an example system that uses DASH to provide *bit rate scalability*—that is, the ability for clients to indicate the bit rate that best suits their capabilities and needs. By far, bit rate scalability is the most common parameter to scale in practice, as network speeds are so variable. The depicted system supports requests for the video at bit rates ranging from 256 kbps to 8 Mbps. Each segment of video is transcoded 6 times in the cloud, producing representations at 256 kbps, 512 kbps, 1 Mbps, 2 Mbps, 4 Mbps, and 8 Mbps.

**Advantages**

Since DASH exclusively uses HTTP, it is inherently a stateless system. This makes it very scalable with respect to the number of active clients, since servers need not track the state of individual clients (Thang et al., 2012). Content served with DASH is easy to deploy to content delivery networks (CDNs), since the server's task is simply to act as a simple file server. This is due to the fact that the more computationally expensive transcoding process takes place before the content is served.

Another advantage of DASH is that it is codec-agnostic. DASH considers the video's representation to be a lower layer of abstraction, and interprets scalability as the process of selecting from a set of pre-transcoded options. The coding parameters and format is a black box. This provides flexibility for engineers to determine the best codec and parameters to use for their specific client base.

**Disadvantages**

One disadvantage of using a system like DASH for scalability is that the scalable parameters must be determined in advance. Clients have no way to provide input in the encoding process; they can only select from a predetermined list of options for each segment of video. Since multi-coding occurs before any requests are made, the range of supported coding parameter values must be selected ahead of time. This can go wrong, for example, if a client requires a video at a bit rate much lower than the ones represented in the pre-computed versions. In this case, the user is likely

to experience buffering events. More importantly, it fundamentally limits smart clients, like vision algorithms, which are capable of providing precise request profiles—such as specifying regions of interest—that cannot be predicted in advance.

Another disadvantage of DASH is that the computational burden of transcoding takes place up front, by the server. If the system has real time constraints, such as if the video content is being live-streamed, then this transcoding process must be performed in real time. Although this results in a system that is scalable with respect to the number of clients requesting the video, it is not scalable with respect to the number of content producers. For this reason, DASH would not scale well in a system that has many more content producers than clients. Furthermore, smaller video distribution systems may not have the available computational resources needed to support DASH for a large number of clients. As the number of overall content producers increases, this trade-off will continue to disadvantage smaller content distributors.

Finally, the fragmented representation of multimedia used by DASH contains a large amount of redundancy, resulting in a huge overhead on the file system. Whereas data compression focuses on eliminating redundant information, DASH takes the opposite approach. This is partially a result of the requirement to provide non-scalable representations to all clients, and partially a result of the decision to reduce per-request server demand. If a layered codec or per-request server computations could be introduced, the file system overhead could be significantly reduced.

## 3.5 On-Demand Transcoding

The final family of approaches for solving the heterogeneous problem are *on-demand transcoding* methods. Whereas scalable codecs and adaptive streaming techniques are characterized by an independent encoding step, which takes place before the video content is served to clients, on-demand transcoding integrates transcoding directly in the request-response lifecycle of the content server. Thus, a single representation of the video is stored on the server and the transcoding process takes place on-the-fly, in response to each request for the content. This has the advantage of providing the most flexibility for clients, and can produce videos that exactly match client bandwidth

and decoding needs while maintaining maximum coding efficiency. The trade-off, however, is that the server system must be capable of transcoding the video in real time.

### 3.5.1 Transcoding

Video *transcoding* is the process of converting pre-coded video content from one representation format to another. A transcoding operation may be *homogeneous*, meaning the video is being transcoded to the same format as the initial format, except with different encoding parameters. This could be used, for example, to transcode a high quality video to a lower resolution, bit rate, or frame rate within the same coding format. Alternatively, transcoding may be *heterogeneous*, meaning the destination format uses a completely different standard than the source format. This could, for instance, prepare some video content to be used by a receiver with specific decoding capabilities, like a DVD player.

There are three transcoding architectures proposed and implemented in the literature: cascaded transcoding, fast transcoding, and guided transcoding (Vetro et al., 2003). These techniques are outlined below.

### 3.5.2 Cascaded Transcoding

The most straightforward approach to designing a video transcoder is to simply cascade an off-the-shelf decoder for the source format with a separate off-the-shelf encoder for the destination format. This is called *cascaded transcoding* because the video content is completely decoded into pixel space before it is completely re-encoded from scratch back to the desired compressed representation. Cascaded transcoding has the benefit of being extremely modular and easy to implement, since it requires no custom coding other than piping the output from the decoder directly into the encoder. However, it comes at the cost of computational inefficiency, particularly when the source and destination formats are very similar. With a cascaded transcoder, the encoder is forced to re-compute prediction and other semantic elements about the video content, even if that information was embedded in the original representation.

Because of the high server computational cost associated with performing a full encode for each client request, cascaded transcoded is rarely used in practice. Use cases are limited to situations where very few requests are expected relative to the amount of content provided.

### 3.5.3 Fast Transcoding

*Fast transcoding* is an alternative to cascaded transcoding in which the transcoder attempts to reduce the time spent encoding by skipping or eliminating certain subroutines of the encoding algorithm (Youn et al., 2000). Standalone video encoders normally need to analyze the source content to generate a semantic understanding of motion and other high-level structures present in the image data. But this process can be computationally expensive. For example, performing an exhaustive motion search across a frame takes time, but yields a rich motion model that describes the motion present in the frame.

Since the encoding portion of a cascaded transcoder is a standalone encoder, the semantic coding decisions including motion search results are calculated from scratch by analyzing the decoded, pixel-domain content. Some of this process may represent duplicated work, especially if similar coding decisions were available as part of the originally coded stream. A *fast transcoder* is a transcoder that uses a modified encoding unit that intentionally bypasses selected subroutines of the encoding process in favor of reusing semantic information from the original encoding (Yuan et al., 2017; Zong-Yi et al., 2013; Peixoto and Izquierdo, 2012; Deknudt et al., 2010).

For example, if both source and destination formats use similar motion compensation algorithms for prediction, then the motion estimation model can be directly transplanted from the source format and reused in the destination format. This way, the transcoder can skip the computationally expensive motion search process entirely. This family of techniques are collectively called “fast” transcoding because the goal is to speed up transcoding time by reusing semantic information from the source, thus reducing or eliminating certain encoding tasks.

### 3.5.4 Guided Transcoding

*Guided transcoding* represents the most recent approach towards on-demand transcoding, in which some of the transcoding work is pre-computed in advance to reduce the per-request computational burden. The pre-computed work is stored on disk on the server as metadata that is completely internal to the transcoding system. The metadata may contain, for instance, pre-computed instructions for which coding decisions to use when transcoding to different bit rates or transcoding to other receiver-driven parameters. When a request for the content arrives, the guided transcoder is directed to the metadata for obtaining the correct encoding decisions instead of deriving them by performing computationally expensive algorithmic searches. This is similar to the way that a fast transcoder works, except that a guided transcoder uses the metadata to deduce the optimal coding decisions instead of the source representation. The goal of guided transcoding is to introduce an intermediate representation to cache some pre-computed results of various transcoding procedures, thereby reducing the per-request burden.

A precursor to guided transcoding was first proposed in 2012 with H.265/HEVC as the base content format (Van Wallendael et al., 2012). The original idea was to store one high-quality representation of the content on the server, together with pre-computed metadata that guides the process of transcoding the representation to other versions at significantly decreased per-request computational complexity.

Since then, three papers have expanded on the original idea. First, (Rusert et al., 2016) investigates ways to reduce the storage requirements of multicast approaches like DASH. The paper notices that in most video content servers, not all content fragments are accessed equally. For instance, older video content is less likely to be accessed, as well as higher layer streams in a scalable, temporal encoding. The authors propose that for rarely accessed content, server disk space can be saved with very little overall computational overhead by implementing guided transcoding. In particular, their guided transcoding approach is to delete the stored residual coefficients from the lower quality, infrequently accessed streams on the server. If the low quality version of a segment is requested for which the residual coefficients have been cleaned up, the server can recompute them

by using the high quality version of the stream, which still exists in full on the server. This can be interpreted as a guided transcode of the high quality stream, using the pre-computed prediction elements as the guiding metadata. The reported experiments used H.265/HEVC as a base codec for the guided transcoding implementation.

Second, in (De Praeter et al., 2017), the same group as (Van Wallendael et al., 2012) extended their original 2012 work by analyzing the performance of guided transcoding when content is transcoded down to significantly lower bit rates than the original, high-quality stream. The paper makes the case that the prediction elements and other encoding decisions made for the high bit rate stream are not necessarily optimal for lower bit rate streams. However, the prediction elements calculated for a particular bit rate tend to be locally optimal—that is, prediction elements can be reused to transcode to nearby bit rates with low compression overhead. The proposed solution is to formulate a series of *control streams* for a given video, each of which contains pre-computed prediction decisions for a different band of bit rates. When a request is made for the content at a particular bit rate, the server simply selects the control stream assigned to that bit rate band. The prediction decisions in that control stream are used to guide the transcode.

Finally, (Hollmann and Sjöberg, 2018) proposes a guided transcoding approach for H.265/HEVC, in which the control streams also contain some residual information to make transcoding easier. In particular, the control streams contain delta coefficients, which are the computed difference between the quantized coefficient values from the high bit rate stream and a low bit rate encoding of the same content. These delta coefficients can guide a transcode operation down to the lower bit rate, because the transcoder can add the delta coefficients to the high quality residual coefficients to reproduce the correct low quality residual coefficient values. The process of creating a control stream is called *deflation* for the purposes of the paper, and the corresponding guided transcoding operation is called *inflation*.

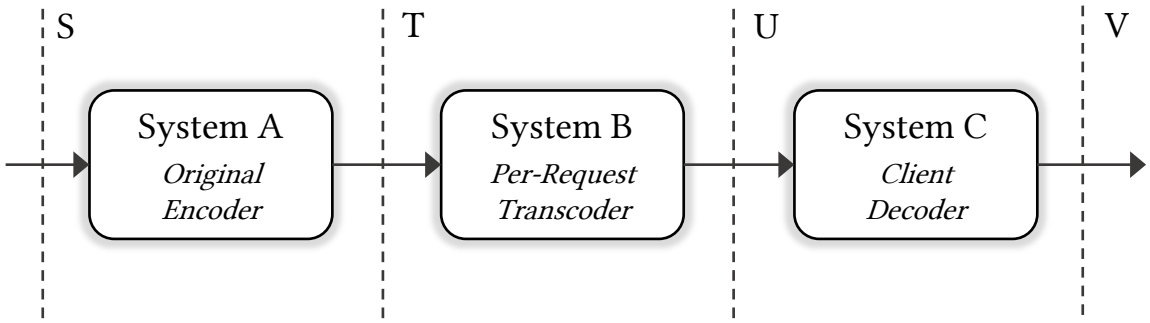



Figure 3.4: This block diagram illustrates a network distributed video coding system, as defined by MPEG in their call for evidence (Group, 2017). The system has four interfaces, $S$, $T$, $U$, and $V$, and three distributed nodes, $A$, $B$, and $C$. Video content flows from left to right, beginning at interface $S$, which represents the source, and ending with interface $V$, which represents the client's decoded video. System $A$ performs a one-time encoding of the content, System $B$ transcodes the video in response to each client request, and System $C$ represents the client decoder.

### 3.5.5 Network Distributed Video Coding

In July 2017, the Motion Picture Experts Group (MPEG) published a technical description of a proposed system that it calls *network distributed video coding* (NDVC). NDVC describes an on-demand video transcoding architecture in which the transcoding process is distributed across multiple nodes in a network to better handle diverse requests (Sjöberg et al., 2017). This paper was released together with a "Call for Evidence on Transcoding for Network Distributed Video Coding" (Group, 2017), which solicited community input about how such a system might be designed. Together, these efforts were intended to draw attention to what MPEG sees as an important growing industry need: a video system architecture with scope comparable to DASH that can solve the heterogeneous client problem through distributed transcoding.

MPEG's call for evidence presents a schematic like the one depicted in Figure 3.4 to describe the parameters of a suitable NDVC system. Three distributed nodes, $A$, $B$, and $C$, are connected by four interfaces, $S$, $T$, $U$, and $V$. Node $A$ receives the input video content from the source through interface $S$ and transcodes it, producing an intermediate representation of the data. This intermediate representation is designed to facilitate per-request transcoding by node $B$, and is stored in interface $T$. When a client requests the video content, node $B$ transcodes this

intermediate representation to formulate a standard-compliant video stream that exactly matches the request's specifications. The transcoded stream is transmitted across interface $U$ to the client device represented by node $C$. Finally, the client decodes the stream, displaying the reproduced video content through interface $V$.

The call for evidence explicitly presents simulcast and full transcoding as diametrically opposed architectural solutions to the heterogeneous client problem, and asks for proposed solutions to provide a better trade-off between them. On one extreme, simulcast solutions like DASH perform all transcoding at node $A$, before any client requests are made. The intermediate representation produced by node $A$ is a collection of fully coded output streams, which are stored on a web server's file system. Node $B$ represents the adaptive streaming server, which responds to client requests by directly reading the appropriate precoded video segments from disk. Thus, in a simulcast system, no per-request transcoding takes place.

On the other extreme, full transcoding approaches perform all transcoding at node $B$, in direct response to each user request. Node $A$ does not alter the initial encoding at all, and the initial stream is therefore also the intermediate representation. When a request is made for the video content, node $B$ transcodes the data just in time to the appropriate format as requested by the client.

A paper releasing the results from MPEG's call for evidence is in pre-print at the time of this dissertation's publication (Praeter et al., 2021). All submissions for the call to evidence used a form of guided transcoding for their NDVC system design, reinforcing its potential (Zhang et al., 2014). Based on the results of the call, MPEG concluded that a NDVC system architecture is a viable approach towards solving the heterogeneous client problem that exhibits real-world benefits over existing simulcast and scalable solutions.

# CHAPTER 4

# Predictive Scalable Coding

As discussed in Chapter 2, many modern video codecs lean heavily on the hybrid predictive-transform coding architecture to achieve data compression. At the same time, Chapter 3 explained how scalable coding is the only class of existing techniques for adaptation that work in the compression layer.

This chapter investigates some of the challenges associated with predictive coding in a receiver-driven, adaptive environment. These challenges are explored through the lens of implementing scalable coding with a hybrid prediction-transform codec. Of the adaptation techniques discussed in Chapter 3, scalable coding is the only approach that works in the compression layer. Scalable codecs offer flexibility by allowing the decoder to make decisions about which subset of content to receive, thereby affecting the availability of prior data. But this poses a challenge: predictive-transform codecs require synchronized decoded picture buffers across the encoder and decoder in order to benefit from predictive coding. Thus, additional system complexity is required to ensure that the decoded buffers at the encoder and decoder remain synchronized despite receiver-driven content selection.

## 4.1 Prediction Drift

### 4.1.1 Availability of Prior Data at the Decoder

Predictive coding requires a decoder to use previously decoded frames of data to anticipate the likelihood of future data values—the decoder uses past data to predict what future data will look

like. In this section, the term "frame of data" represents an abstract data unit, which may refer to an entire video frame, a single macroblock, a coding unit, a single symbol, etc.

Often, the process of forming a prediction for the next frame of data will be directed by the encoder through *prediction syntax elements*, encoded as part of the compressed data stream. Prediction syntax elements represent instructions passed from the encoder to the decoder, which tell the decoder how to form a prediction. This assumes that the encoder and decoder are in agreement about the recovered values from prior signaled frames that are being used as the basis for prediction. It also assumes that the decoder has recovered and retained the prior signaled frames. These assumptions may not always hold true; for example, consider the following situations.

- **Channel errors** – A channel error during storage or transfer of the compressed bit stream may cause one or more transferred bits to flip, resulting in incorrectly decoded values. In this situation, prior frames may have been decoded, but are not accurate with respect to the encoder's expectation.

- **Unreceived data** – The client simply did not receive or decode the prior data. Perhaps the client skipped to the middle of the compressed bit stream and is trying to decode the latest data frames without having received earlier frames. The client will be unable to decode or interpret any predictive-coded frames that are signaled by referencing previous frames.

- **Deleted data** – The client had previously received, decoded, and recovered earlier data frames, but did not retain them in memory. Alternatively, perhaps a memory corruption in the decoded picture buffer affected the integrity of the reconstructed data. In any case, the necessary decoded prior frames are either completely unavailable or are no longer suitable for prediction.

- **Client-driven decoding** – The client is making its own choices about which data to receive, and the encoder does not know which portions of the data frames have been decoded and which have been ignored. Thus, the encoder and decoder are not synchronized with respect to their knowledge of prior frames.

In each of these situations, the decoder is unable to faithfully decode future frames that are represented using predictive coding techniques. In particular, this dissertation investigates how predictive coding can be adapted to work in the last situation described above, where the receiver makes the decisions about which data elements to receive.

### 4.1.2 Recursive Predictive Structure

Predictive coding can be interpreted as the process of generating a recursive representation for a given data signal (Ohm, 2005). With this characterization, the first data frame is encoded independently (i.e. the "base case"), and is followed by a series of recursive, dependent representations of the subsequent frames, which are defined in terms of previously coded frames. As with all recursively-defined data, care must be taken to maintain fidelity at every iteration. This is because any error or missing data in an early frame will propagate forward to all recursively-defined dependent frames. Furthermore, the effect of an error is amplified the more times it is reproduced by the recursive algorithm.

### 4.1.3 Prediction Loops in Hybrid Codecs

Whenever predictive coding is used by a data compression algorithm, a recursive coding dependency called a *prediction loop* is introduced into the system. Prediction loops are characterized by the recursive structure described in the previous section. They represent a feedback loop in the encoded data such that future coded data frames are represented in relation to prior coded data frames. Casting the task of data compression as a recursive definition problem in this way can produce highly sophisticated human-engineered codecs. However, every prediction loop introduced in the system is a potential point of failure. Any incomplete or missing information at the decoder that was expected by the predictive encoder results in reconstruction error that is amplified as decoding continues. In the worst case, such a situation can result in fatal decoding errors where the decoder misunderstands the state of the stream and is no longer able to interpret the encoded symbols due to the missing information.

The dominant structure for hybrid video codecs in use today contains two prediction loops: one for intra- and inter-frame prediction, and one for entropy context adaptation. These two prediction loops are illustrated in the schematic depicted by Figure 4.1, and are also described in the sections below.

**Predictive-Transform Loop**

The "prediction" part of the hybrid "predictive-transform" video codec design forms the first prediction loop. Modern hybrid coders support both inter-frame and intra-frame prediction techniques.

- **Inter-frame prediction** refers to motion estimation and compensation. It is the process of encoding motion vectors as part of the compressed representation to describe how a decoder should copy patches of reconstructed pixels from prior video frames to produce a prediction of the pixel values in the next video frame.
- **Intra-frame prediction** refers to textural-based prediction, in which the reconstructed pixel values from a previously-coded neighboring coding block are "smeared" across the next coding block in the same frame to form a prediction.

See Chapter 2 for a richer description of both inter-frame and intra-frame prediction.

As the decoder decodes the compressed stream and reproduces video frames, it stores each reconstructed frame in a *decoded picture buffer* (DPB) for use in future prediction. Both inter- and intra-frame prediction use the data in the DPB to predict future pixel values. The encoder must therefore also compute and maintain this same DPB so it can express future frames in terms of prior ones.

Note that the predictive-transform loop also represents advanced motion vector prediction (AMVP) and other newer techniques that involve predicting the prediction syntax elements for a frame based on prediction elements signaled as part of previous frames.

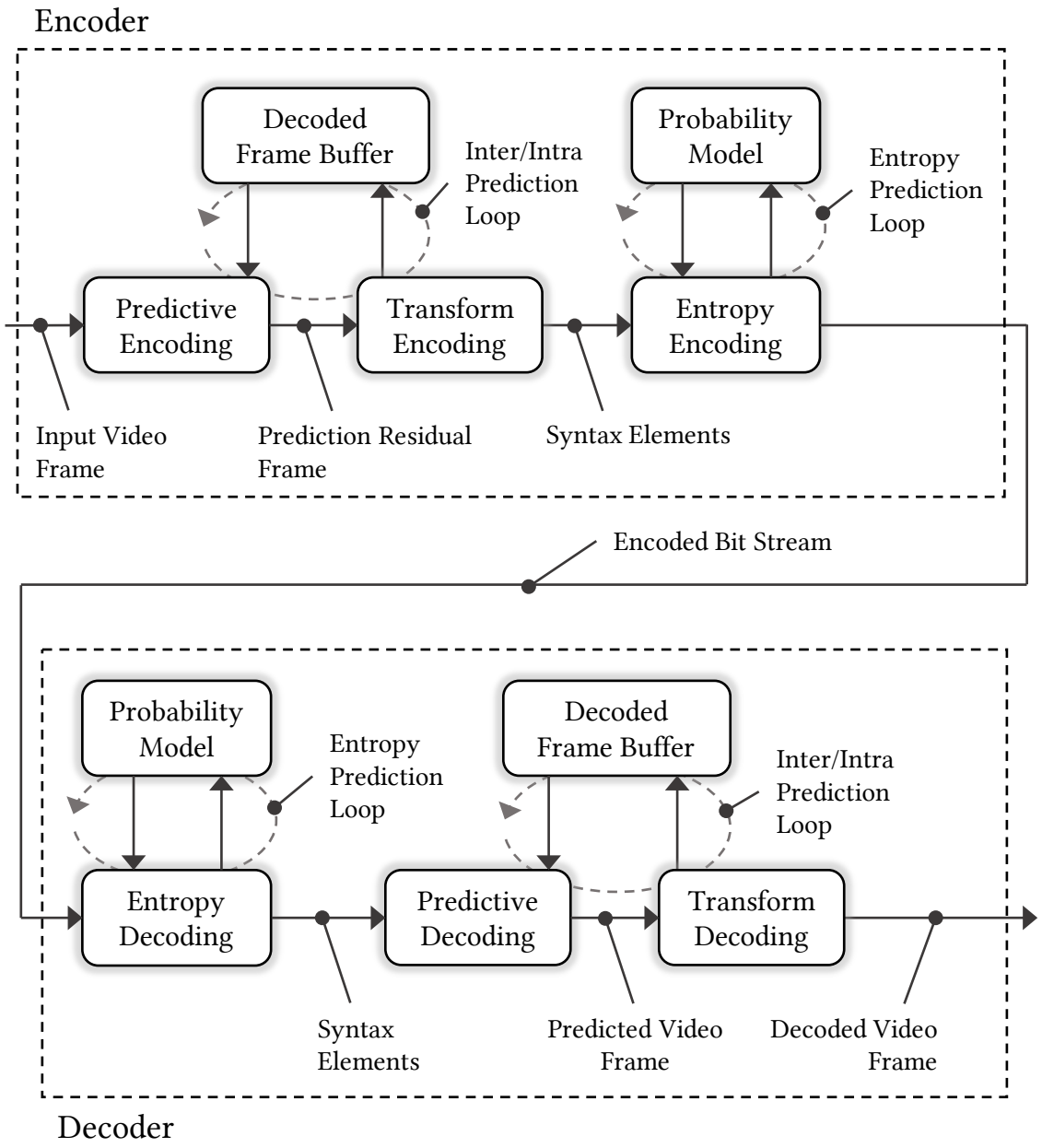



Figure 4.1: This schematic outlines a simplified hybrid video codec architecture and highlights the prediction loops embedded in the system. Two prediction loops are depicted: one for predictive-transform coding, labeled "Inter/Intra Prediction Loop," and one for entropy coding, labeled "Entropy Prediction Loop." To avoid drift in the decoded reproduction, both prediction loops must remain synchronized across the encoder and decoder.

Any discrepancy between the encoder and decoder's copy of the decoded picture buffer may be a source of error in the decoder's reconstruction of the compressed video frames. Furthermore, any such error may be propagated forward by either inter-frame or intra-frame prediction. Note that intra-frame prediction is only capable of propagating error in spatial dimensions, whereas inter-frame prediction is only capable of propagating error in the temporal dimension.

### Context-Adaptive Entropy Coding Loop

Once prediction and transform of a video frame is complete, the next step is to encode the resulting prediction and residual syntax elements to the output bit stream. Modern video codecs ultimately use entropy coding techniques for this. In particular, a common approach is to use a context-adaptive binary arithmetic coding (CABAC) algorithm to break down and encode block split decisions, motion vectors, quantized coefficients, etc as a series of binary decisions. This process is described in more detail in Chapter 3.

Arithmetic coding algorithms use a context-adaptive probability model for encoding, which is essentially an estimated probability density function (PDF) for the value of the next encoded symbol. This PDF is improved, or *adapted*, as symbols are encoded, based on the frequency of prior symbols. In other words, the context model used by arithmetic coding algorithms like CABAC represents a prediction for the value of the next encoded symbol.

Much like the decoded picture buffer, the adaptive context model used by CABAC must be synchronized between the encoder and the decoder. Any discrepancy between these context models will result in a bit stream of encoded symbols that is seemingly nonsense to the decoder. This is because the bit pattern of the encoded symbols is entirely dependent on the context model.

An error in the entropy coding prediction loop is often more serious than an error in the predictive-transform prediction loop. Discrepancies in the decoded picture buffer simply result in the propagation of incorrect pixel values either spatially or temporally. However, a discrepancy in the entropy coding probability model results in the permanent divergence of encoded and decoded symbols.

### 4.1.4 Decoder Drift

When missing or corrupt data results in mismatched predictions between an encoder and a decoder, the decoder's reconstructed pixel values drift away from the true source values. As playback time elapses, motion compensation (i.e. temporal prediction) causes these reconstruction errors to propagate forward to future frames. This behavior is called *decoder drift*. If left unattended, decoder drift can quickly become noticeable as video quality rapidly decays.

#### Instantaneous Decoder Refresh (IDR) Points

Decoder drift is not entirely unexpected in the realm of video playback. To reduce the potential long-term effects of drift on a video stream due to temporal propagation of reconstruction errors, encoders will typically intentionally incorporate regular *instantaneous decoder refresh* (IDR) points in an encoded video stream. An IDR point is an instant in playback time across which no temporal prediction is allowed. Once an IDR is reached, a decoder is free to clear its decoded picture buffer. Frames coded after an IDR are guaranteed not to reference frames that were coded before the IDR point.

IDR points are useful for mitigating drift, but they are also helpful for decoders interested in seeking to a particular time in the middle of the stream without having to decode starting at the beginning. To seek to a particular playback point in the stream, a video player must simply identify and seek to the location of the nearest IDR encoded in the bit stream before the instant at which playback should start. The player must then decode the stream beginning from the IDR point, and progressing forward to the desired starting point.

## 4.2 Challenges

As discussed in Chapter 2, predictive coding is an integral part of hybrid video codec design. However, Chapter 3 made the case that support for scalable video coding is all but necessary

to address the heterogeneous client problem. This section describes the challenges associated with integrating predictive and scalable coding in the same codec.

At face value, scalable coding and predictive coding seem to have opposing goals. On one hand, scalable codecs are designed so that decoders can unilaterally and dynamically choose which subset of the overall data to receive. On the other, predictive coding only works if the encoder and decoder are in precise agreement about what prior data has been received. To illustrate this tension, consider inter-frame (i.e. motion-based) prediction. To predict the next frame, a synchronized, previously encoded frame must be available for reference. However, with a scalable codec, decoders may have differing versions of the same prior frame, each with different SNR qualities, bit depths, resolutions, etc. If there is no agreement between encoder and decoders about the prior frame, synchronized prediction becomes difficult. And this challenge is not unique to inter-frame prediction. The other examples of prediction loops in video coding must be addressed in a scalable environment, too. Care must be taken to ensure that the context models used by context-adaptive arithmetic coding are adapted and reset appropriately according to the symbols received by the decoder, and that intra-frame prediction remains synchronized across block boundaries.

The tension between synchronized prediction and scalable decoding can be solved in two ways: either by working around the prediction loops to ensure that the encoder and decoder remain synchronized despite the introduction of scalable coding, or by permitting the decoder to drift away from the prediction expected by the encoder as scalable decisions are made. These two approaches are elaborated below.

### 4.2.1 Drift-Free Scalable Predictive Coding

If prediction synchronization between the encoder and decoder is to be maintained in a receiver-driven or otherwise scalable environment, then the encoder must either track, anticipate, or be told which subset of data was received and decoded by the client. Depending on the system architecture and codec design, a solution might take any of the following forms.

### Avoiding Drift by Transcoding

With a just-in-time transcoding architecture, it is possible for the encoder to deduce which information is received by the decoder based on which subset(s) of the stream were explicitly requested and transferred to the client. If there are multiple clients, the transcoder can maintain the custom prediction data for each client in real time according to which data that client has received.

This approach could potentially achieve theoretically optimal rate-distortion performance, since a full, custom transcode of the source content is performed for each client. The encoder can maintain full knowledge of all data previously received by the decoder, and can encode future frames accordingly. However, producing an optimal representation of the requested data for each client comes at the cost of significantly increased per-client encoder complexity.

### Avoiding Drift by Multiple Layer Prediction

With a layered codec architecture, the encoder and decoder could retain separate synchronized prediction reference data for each layer. In this scheme, all layers can utilize predictive coding; however, to stay synchronized, no layer's prediction can incorporate data received from another layer unless it is dependent on that layer in the codec's dependency graph (see Figure 3.1). This ensures that inter-layer coding dependencies are respected. Although this requires careful dependency management and a more complex codec, this is the strategy adopted by H.265/HEVC's scalable extension SHVC.

Unfortunately, despite allowing prediction at each layer, theoretically optimal rate-distortion performance is unattainable with this approach. When predictive coding is used to signal data in the lowest layers, it cannot reference previously coded available data from higher layers to form a prediction. Any correlation between a lower layer and a higher layer cannot be exploited by lower layers. As a result, the predictions produced by lower layers are suboptimal when compared to what would have been possible if higher layer data could be referenced for prediction. This problem is exacerbated for decoders that have received most or all of the available layers. The decoder cannot use its high quality reproduction for prediction in lower levels, and must instead maintain

separate low quality reproductions to reference when predicting for these levels. Therefore, the rate-distortion performance penalty of this approach is worse when decoding at higher bit rates.

#### Avoiding Drift by Base Layer Prediction

Finally, an extreme case of the layered approach above is to limit predictive coding across all layers so that only information received in the base layer can be referenced. This approach is used, for example, by MPEG-4's *Fine Granularity Scalability* (FGS) mode, which does not use inter-frame prediction in enhancement layers at all. Another example is H.264/AVC's scalable extension SVC, which requires only a single decoding loop. Since all decoders are guaranteed to receive the base layer, they will all have that data available for prediction. This simplifies decoder architecture so that only one prediction reference buffer is needed, regardless of the number of layers provided. Rate-distortion performance is reduced, however, for the same reason as before: clients who choose to decode enhancement layers are not allowed to incorporate the additional data in the prediction loop lest they desynchronize from the encoder. Predictions are therefore less accurate than they would be if the complete available data could be referenced. Ultimately, this means extra prediction residual data will be transferred unnecessarily.

### 4.2.2 Drift-Controlled Scalable Prediction

The approaches in the previous section address drift at the expense of additional computational complexity at the encoder and decoder. Furthermore, the layered approaches result in suboptimal rate-distortion performance when higher layers are incorporated in the reconstruction.

If these limitations are unsuitable, an alternate approach is to simply allow drift to be introduced in the system as the decoder scales the stream. In an attempt to form an optimal prediction given available data, the decoder will use the highest quality reproduction available as its reference every time it is directed to form a prediction. In a scalable system, not all decoders have access to the full quality reference. Prediction quality will therefore vary by decoder, yielding reconstructions of different quality (i.e. drift).

Suppose original frame $F_i$ is received by scalable decoder $j$, which produces reconstructed frame $F_{i,j}$. Due to discrepancies already present in the prediction loop, $F_i \neq F_{i,j}$. Thus, the amount of drift experienced by decoder $j$ for frame $F_i$ can be quantified as

$$D_{i,j} = F_i - F_{i,j} \tag{4.1}$$

As decoding continues, temporal prediction causes the drift to be propagated forward, eventually potentially dominating the signal. Additional mechanisms can therefore be added to the encoder to control—but not entirely eliminate—the drift experienced by decoders.

One option is that the encoder can track or estimate the maximal amount of drift that may be experienced by decoders receiving different subsets of the stream. Drift tracking at the encoder can be implemented by following various scalable decoding paths and measuring the resulting drift directly. If drift gets too high along any one path, the encoder can incorporate an IDR to reset the prediction loop for all decoders. For example, suppose a encoder will permit drift up to a certain PSNR threshold, $d_{\text{threshold}}$. After frame $i$ is encoded, the encoder computes $d_{i,\max}$, the maximum drift experienced by any of the decoder paths for frame $i$.

$$d_{i,\max} = \min_j \left(\text{PSNR}\left(D_{i,j}\right)\right) \tag{4.2}$$

As long as $d_{i,\max} > d_{\text{threshold}}$, encoding can continue normally. Once $d_{i,\max} \leq d_{\text{threshold}}$, an IDR is prepared for the next possible frame which resets all decoder prediction loops.

In another approach explored by Chapter 6, the encoder can attempt to compensate for the drift experienced by the decoder. The goal is to manage drift equally across all possible decoding paths. As before, the encoder tracks or estimates the drift along various decoding paths. However, the key realization for this approach is that an encoder can reduce or even eliminate drift along any one decoding path $j$ by subtracting $D_{i,j}$ from the prediction residual signal coded in the output stream for frame $F_i$. In other words, if the encoder knows the exact amount of prediction drift experienced by a decoder, it can compensate for the drift in the residual.

Thus, an encoder that is actively tracing various decoder paths can take turns compensating for the drift experienced by each path by altering the residual signal accordingly. Note that this is an optimization problem where adjusting for drift in one decoding path may adversely affect other decoding paths. Therefore, encoders may need to prioritize certain decoding paths over others or try to find a balance between each path. For example, an encoder might choose to prioritize the decoding path associated with a certain bit rate by continually compensating for drift in that path. However, other decoding paths—particularly those with significantly different coding parameters or bit rates—will suffer in terms of rate-distortion performance.

Instead of compensating for drift equally among all decoder paths, the encoder could identify a certain decoder path and prioritize that path over others for drift compensation. This could be used in a situation where an encoder knows in advance to optimize for a specific range of bit rates. The encoder would specifically track and compensate for drift in that range, resulting in near-optimal rate-distortion performance for decoders at that target bit rate. Decoders receiving at bit rates far away from the target range would experience more drift and lower efficiency. Reduced efficiency and drift would be experienced at both extremes, including decoders at higher or lower bit rates.

A final approach attempts to limit the influence of drift over time. The idea is to introduce a decay factor $a \in [0, 1]$, which is used by the decoder to dampen all prediction values. For example, after a motion vector is applied to obtain a motion-compensated block of pixels, the decoder multiplies the predicted intensities by $a$ before processing the residual signal. As playback time elapses, the influence of drift from older frames will therefore decay exponentially. By adjusting the value of $a$, the amount of decay applied can be controlled. $a$ can either be set by the encoder or assumed by the coding format itself. However, the encoder must be aware of the decay factor so that it can compensate for the lost information by adjusting the residual signal in response. Ultimately, this is a heavy-handed approach that trades-off rate-distortion performance by reducing the overall effect of prediction in order to stabilize drift.

### 4.2.3 Effect of Scalable Coding on Rate-Distortion Performance

The rate-distortion performance of a scalable codec is typically assessed by creating a single scalable encoding of a given video and plotting the R-D curve that results from decoding different subsets of the stream. This scalable R-D curve is then compared against a non-scalable R-D curve for the same video. The non-scalable curve is produced by encoding multiple independent non-scalable versions of the same video content as before at different target bit rates. For each non-scalable encoding, a point is plotted indicating the bit rate and the achieved quality. It is typical for a non-scalable R-D curve to outperform the scalable alternative. However, based on which scalable technique is used, the performance of a scalable encoding can approach that of a non-scalable coding, particularly for certain target bit rate ranges.

With the exception of the just-in-time transcoding architecture, all approaches discussed above for designing a scalable codec with predictive coding will result in suboptimal rate-distortion performance when compared with a non-scalable encoding at the same bit rate.

- The drift-free, layered approaches used by scalable codecs including SVC and SHVC produce an R-D curve that is anchored with the highest rate-distortion performance at the base layer. This is because the base layer is usually a non-scalable encoding itself. As the number of decoded layers increases, rate-distortion performance suffers due to the problem of artificially limited lower-layer prediction. Thus, clients receiving at high bit rates experience the lowest rate-distortion performance.

- If drift is simply allowed by the encoder with no control mechanism, the highest bit rates experience the best rate-distortion performance. Lower bit rate decoders suffer from suboptimal prediction, and are therefore susceptible to drift.

- The drift-tolerant approaches allow the encoder to specify a specific bit rate to prioritize, or optimize across all bit rates equally. Drift will always cause suboptimal rate-distortion performance for these family of techniques, but the encoder is able to direct the effect—and the penalty—away from certain bit rates by applying drift compensation.

Figure 4.2 sketches the shape of the R-D curves for each scalable approach above, assuming that each approach is applied to the same source video content. The figure illustrates how each technique's R-D curve prioritizes a different range of bit rates, but no technique is everywhere optimal.

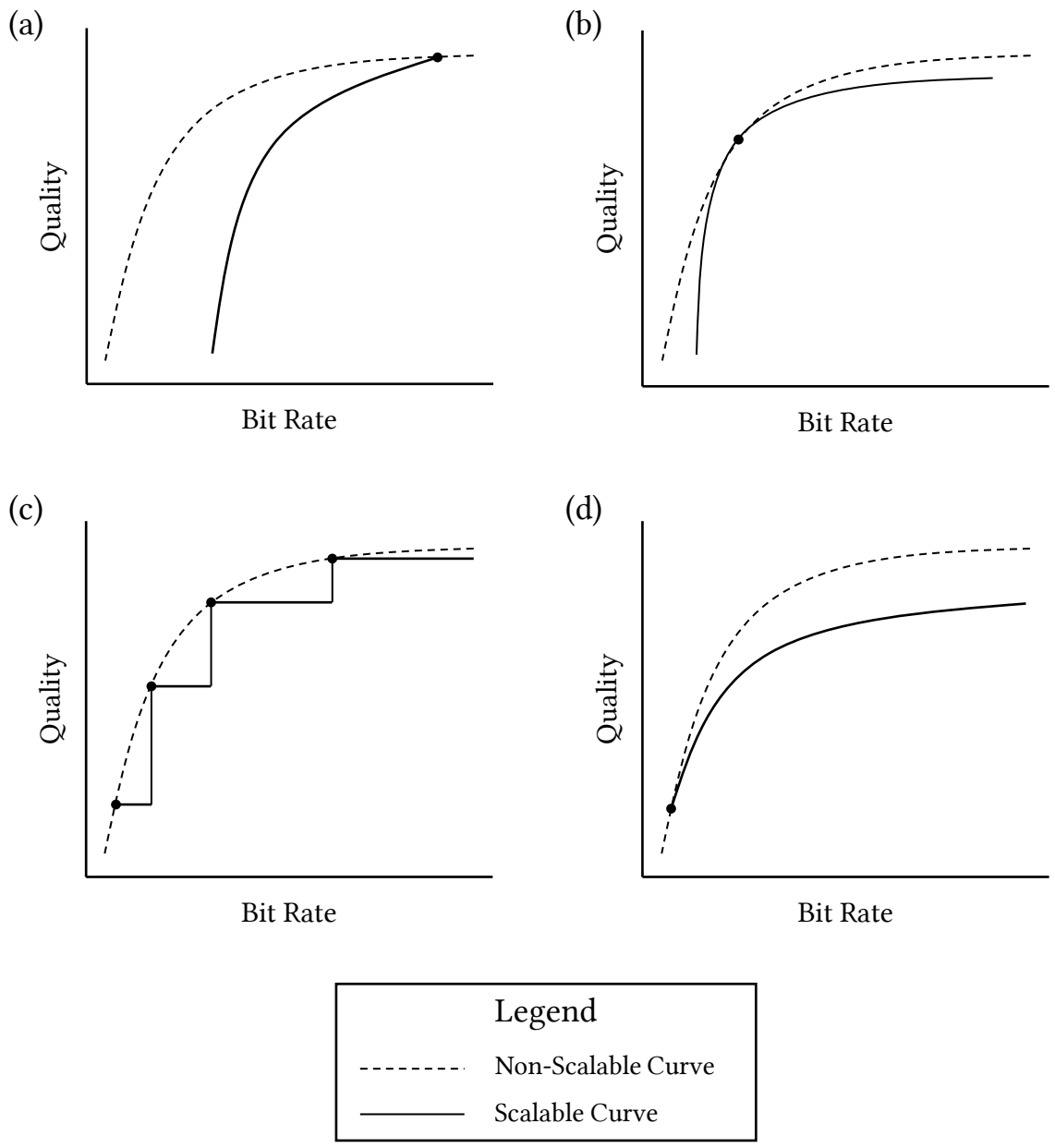



Figure 4.2: These R-D curves illustrate how different scalable approaches affect rate-distortion performance when applied to the same video content. Each approach is compared against a non-scalable encoded R-D curve. Subfigure (a) represents the case where the transcoder allows the decoder to drift freely. Subfigure (b) illustrates a drift control mechanism anchored at a particular bit rate. Subfigure (c) shows a simulcast/adaptive bitrate streaming solution. Subfigure (d) depicts a traditional bottom-up layered coding approach.

# CHAPTER 5

## Content-Adaptive Entropy Coding

Chapter 3 outlined the three main classes of video adaptation approaches: scalable coding, adaptive streaming, and on-demand transcoding. These strategies are computationally expensive, do not scale well with the number of videos produced, and throw away important semantic information about the captured scene that is embedded in the initial encoding. The discussion concluded by citing a recent call-for-evidence published by MPEG, outlining the need for new adaptation systems. In particular, MPEG claims that new systems should take a more flexible, distributed approach towards adaptation.

The problem with existing strategies is that they carry out full transcoding of the initial representation, and do not allow receivers to actively guide the process. Every time a video is transcoded to and from pixel space using a lossy codec, quality is lost. More significantly, the semantic predictive information embedded in the prior encoding is forgotten and recomputed by performing a computationally demanding prediction search. This results in high computational costs for video distributors, adapted video output that is not flexible to receiver needs, and endpoint content that was likely transcoded so many times that it no longer semantically reflects the initial representation.

To solve these challenges, adaptation must be fundamentally recharacterized. The flaws exposed by existing techniques can be avoided by designing adaptation methods that sidestep full transcoding entirely; instead, they should reuse the source encoding as much as possible. This has two benefits: first, it greatly reduces the computational burden of re-encoding, and second, it results in a system that adapts downward from the highest quality representation. Furthermore, new adaptive solutions must be more flexible for the receiver than existing approaches (i.e. they

should be "receiver-driven"). This ensures that the diverse, sophisticated applications of the future will be well-supported by these new adaptive techniques, even if their application needs are not yet understood.

To that end, this chapter proposes a novel, fundamental approach towards top-down adaptation that fits the prescription. The proposed technique adapts any data source that can be modeled abstractly as a sequence of symbol tokens. Instead of adapting by remodeling the source encoding, adaptation is achieved by selectively dropping some symbols from the original stream during entropy coding. The decision of which symbols to drop is exclusively determined by a rate controller algorithm specified by the receiver. By supplying the rate controller, receivers can effectively pick and choose which encoded symbols to receive from the stream. There are myriad reasons why a smart receiver may not wish to receive every symbol. For instance, perhaps not enough network bandwidth is available to receive the whole stream, and the receiver must make choices about which symbols to receive and which cannot be afforded. Alternatively, perhaps the receiver is only interested in a certain subset of the encoded symbols, based on the application's needs. The salient point is that as receivers become more sophisticated, they can be expected to have a much richer understanding of their available computing resources and application needs than the provider does. Distributing these decisions to the receiver about which symbols to keep promises to be a more flexible approach, allowing the same source content to be dynamically adapted very differently for diverse smart applications.

We term the proposed technique *content-adaptive entropy coding* because it adapts the source content to each receiver's needs during entropy coding. For evaluation, two implementations of content-adaptive entropy coding with video data are presented. First, content-adaptive entropy coding is successfully applied to an H.265/HEVC coded symbol stream, demonstrating how top-down, layered adaptation is possible with predictive-coded video data. Next, it is also applied to a M-JPEG coded symbol stream to demonstrate how a receiver-driven approach can outperform traditional cascaded transcoding for certain use cases.

## 5.1 Formulation

This section presents the novel content-adaptive entropy coding algorithm that embodies a major contribution of this dissertation. The proposed scheme takes as input an arbitrary, source-modeled sequence of abstract symbols $S = (s_0, s_1, s_2, \ldots)$. Let $s_i \in A$ be the $i$th symbol value from the input stream $S$, where $A$ represents the symbol alphabet. Ultimately, these symbols are encoded to the compressed bit stream using a standard arithmetic coding algorithm. To produce code words of the correct length, arithmetic coding works by assuming the encoder and decoder share a common estimate of the probabilities of each symbol value $a \in A$. Therefore, both decoder and encoder must maintain synchronized copies of an adaptive probability model $PM : A \rightarrow [0, 1]$. Before each symbol $s_i$ is encoded, the model estimates the probability distribution for the symbol's value.

$$PM(a) \approx P(a = s_i) \tag{5.1}$$

Now, suppose an unaltered context-adaptive arithmetic entropy encoder and decoder exists with functions for encoding and decoding symbols to and from the compressed stream. Let $C$ represent the compressed stream. Suppose $\texttt{encode}(C, PM, s_i)$ encodes symbol $s_i$ to the compressed stream $C$ using probability model $PM$, and $\texttt{decode}(C, PM)$ decodes and returns the next symbol from the compressed stream $C$ using probability model $PM$. Since the codec is context-adaptive, an $\texttt{update}(PM, s_i)$ routine is also implemented which adapts the probability model $PM$ to incorporate the fact that symbol $s_i$ has been coded. After the update routine is called for symbol $s_i$, it is assumed that $PM$ has been adjusted to model the estimated probabilities of the next symbol $s_{i+1}$. Naturally, the encoder should only update its copy of the probability model $PM$ after a symbol is encoded, and the decoder should update only after one is decoded.

So far, this notation is sufficient for modeling an unaltered context-adaptive arithmetic entropy encoder that does not support content adaptation on its own. The key to incorporating content adaptation is to introduce a synchronized rate controller algorithm `RC` which is reproduced at both the encoder and decoder. `RC` determines in advance whether enough bandwidth exists to

encode symbol $s_i$ in the bit stream, returning a binary indicator value as output. `RC` must be executed by the encoder before encoding symbol $s_i$, and by the decoder before decoding it. If `RC` determines that enough bandwidth exists (i.e. it returns `true`), then the symbol $s_i$ is encoded/decoded like normal. If not enough bandwidth exists, then symbol $s_i$ cannot be encoded or decoded; we say that the symbol is *dropped* from the output bit stream. This is the mechanism by which content is adapted.

Regardless of whether symbol $s_i$ is encoded or dropped, the probability model $PM$ must be updated at both the encoder and decoder to model the probabilities of $s_{i+1}$. If the rate controller determined that symbol $s_i$ should be encoded, then $PM$ can be updated like usual using the $\texttt{update}(PM, s_i)$ routine. However, if $s_i$ was dropped, $PM$ must be updated by both the encoder and decoder without using knowledge of the value of $s_i$. This is because the decoder did not receive the value of $s_i$ and therefore cannot use it. To handle this case, the content-adaptive formulation of entropy coding adds an overloaded version of the update function, $\texttt{update}(PM)$. This version of the function does not take into account the value of symbol $s_i$, but still updates the probability model to model probabilities for the next symbol $s_{i+1}$.

This also highlights an important strength of content-adaptive entropy coding: even when the decoder does not recover a symbol value, it still is aware of the fact that the symbol value exists but is missing. Some decoders may choose to make a educated "guess" about the value of the missing symbol, based on its probability model $PM$. For instance, the decoder could always fill in the value of missing symbols using the most probable symbol (MPS). Alternatively, a stochastic regime could be implemented where the missing symbol's value is stochastically inferred according to the estimated probability distribution from $PM$.

The rate controller algorithm `RC` must be careful to make its decision only based on information available at the decoder. For instance, `RC` cannot use the value of $s_i$ to make its decision, since the decoder does not have access to that value yet. However, `RC` can use the state of the probability model $PM$, since that is available at the decoder (i.e. $\texttt{RC}(PM)$). Moreover, `RC` can incorporate sophisticated context awareness to help make its decision. For instance, some symbols

may be deemed extra important due to their semantic meaning within the larger stream. Alternatively, a specific receiver may have a particular interest in certain symbols, and can incorporate this information in the design of RC.

The exact RC algorithm for making symbol drop decisions may be application-specific, or even receiver-specific. By supplying the RC algorithm to the encoder, a receiver could effectively drive adaptation. For example, if a particular receiver needs a low bandwidth version of the content, it could supply a suitable RC that liberally drops symbols from the source stream. Alternatively, the video provider may design and provide a few predefined, off-the-shelf rate controller algorithms which implement common use cases among receivers. Instead of supplying the rate controller, some receivers may instead select a premade one from this list. This also opens the possibility of caching the streams produced by premade rate controllers, thereby increasing server-side performance for the most common use cases.

## 5.2 Budgeted Rate Controller

Clearly, designing a "good" rate controller algorithm RC that matches the receiver's application needs is integral to a successful implementation of content-adaptive entropy coding. Exposing the design and implementation of such an algorithm to the receiver is an appropriate layer of abstraction, because it gives future applications the flexibility and control needed to fully adapt the encoded content to their unique use cases. The design of sophisticated rate controllers tailored for specific use cases therefore is a promising avenue of future research. This dissertation initiates the process by proposing a simple—yet powerful—formulation of a bit-budgeted rate controller.

The rate controller proposed in this section will be used by all implementations in this chapter. It is defined by function $\texttt{RC} = \delta(b, PM)$, which takes the following two input parameters, and returns a binary indicator decision:

1. A "bit budget" $b \in \mathbb{R}$ representing the highest acceptable number of bits to use when coding the symbol.

2. The current state of the probability model $PM$ which would be used by the arithmetic coder if the symbol is ultimately encoded.

The strategy for $\delta(b, PM)$ is to use $PM$ to identify the worst-case number of bits that could possibly be required to encode the symbol if the least probable value is signaled. Next, the available budget $b$ is checked to see if it contains enough bits to encode that value. If enough bits are available, then the symbol can be encoded; otherwise, it cannot.

The proposed bit-budgeted rate controller indicator function $\delta$ is given by

$$\delta\big(b,\ PM\big) = \begin{cases} 1 & \text{if } b \geq -\log\Big(PM\big(\texttt{LPS}(PM)\big)\Big) \\ 0 & \text{otherwise} \end{cases} \tag{5.2}$$

where `LPS` is a function that extracts the least probable symbol of probability model $PM$, and $\delta(b, PM) = 1$ indicates that the symbol should be encoded in the layer. At a high level, $\delta(b, PM)$ reduces the question of whether to encode or drop each symbol down to the act of selecting an appropriate value of $b$ for the symbol. In essence, a symbol is only encoded if enough bits are available in the bit budget $b$ to encode the least probable symbol (i.e. the most expensive symbol).

While fundamentally simple, this formulation for rate control is quite powerful. As later experiments will show, selecting $b$ appropriately for each symbol can successfully support targeted, receiver-driven use cases such as region-of-interest selection. The scheme for determining the value of $b$ could be as simple as a fixed allotment for each symbol, or it could adjust based on an understanding or approximation of the importance of each symbol that is being coded. In general, higher values of $b$ should be used for more important symbols, as this increases the chance that they will be coded. Some streams may contain certain symbols which have maximal importance, in the sense that their value is required at the decoder to determine the meaning or presence of future symbols. This is an artifact of tight inter-dependence between subsequent symbols used in many existing codecs. In essence, such symbols must be encoded to ensure that they will be received by

the decoder. This situation is handled by setting the bit budget for these symbols to infinity, which forces $\delta(\infty, PM) = 1$ to be true as desired.

### 5.3 Layered Scalable Extension

So far, content-adaptive entropy coding has been introduced as a dynamic adaptive technique in which a smart receiver supplies or selects a rate controller algorithm, the provider generates the adapted encoding, and the result is streamed to the receiver. This is anticipated to be the primary use for the proposed technology. However, content-adaptive entropy coding can also be used to produce a layered scalable codec. Recall from Chapter 3 that layered scalable codecs partition the encoded data into a base layer and a series of enhancement layers. Content-adaptive entropy coding elicits this behavior by treating its output stream as the base layer, and creating one or more enhancement layers that signal the symbols which were originally dropped by the base layer's rate controller. In other words, the symbols that were dropped from the base layer constitute another symbol stream which can be input to a second instance of a content-adaptive entropy coder. The second instance therefore produces another encoded bit stream which serves as the first enhancement layer. Any symbols dropped from this layer constitute yet another symbol stream which can be encoded by a third instance to produce a second enhancement layer. This process can be repeated to induce as many enhancement layers as necessary.

To formalize this idea, suppose an arbitrary input symbol stream $S = (s_0, s_1, s_2, \ldots)$ is to be coded into $k > 0$ layers, numbered from $L_0$ to $L_{k-1}$. Each layer represents an entropy coded bit stream of symbols encoded using a standard arithmetic coder, just as before. However, this time, each layer maintains its own separate probability model for coding symbols. Define the probability model for layer $L_j$ to be $PM_j$. To maintain dependencies across layers, $PM_j$ can be context-adaptive, but must only incorporate knowledge of symbols coded in layers $L_0$ to $L_j$, inclusive. This is so that a decoder which receives only layers $L_0$ to $L_j$ ($0 < j < k - 1$) could still maintain the correct corresponding probability models $PM_0$ through $PM_j$. In practice, this means that probability models at higher layers will be able to incorporate more prior symbol knowledge

into its estimates for coding future symbols. Thus, higher layers can be expected to achieve better compression rates than lower layers.

### 5.3.1 Encoding a Symbol

Each input symbol $s_i$ from source stream $S$ is coded into at most one layer. Suppose $s_i$ is the next symbol to be coded. To determine which layer contains symbol $s_i$, every layer implements its own rate controller algorithm. Let $RC_j$ represent the rate controller algorithm for layer $L_j$. Just like in the non-layered case, the rate controller algorithms may be supplied by a smart receiver or they may be predetermined by the encoder. To determine which layer contains symbol $s_i$, the rate controllers are sequentially evaluated, beginning with $RC_0$ and proceeding upwards until either a layer's rate controller elects to include the symbol in its bit stream, or the final rate controller $RC_{k-1}$ elects not to code the symbol. Once a symbol is included in a particular layer, higher layers need not evaluate their indicator decisions for the current symbol, because the target layer has already been identified. If no layers elect to include the symbol, then it is dropped from the adapted layered encoded stream.

Suppose the target layer for symbol $s_i$ is determined to be layer $L_j$. The arithmetic coder therefore uses probability model $PM_j$ to encode/decode symbol $s_i$ to and from bit stream $L_j$. After the symbol is encoded, the probability models for layers at or above $L_j$ are updated by the encoder to reflect the fact that $s_i$ was signaled (i.e. probability models $PM_j$, $PM_{j+1}$, ..., $PM_k$ are updated). Decoders that receive these higher layers must update their copies of the probability models in the same way. Additionally, the probability models for layers below $L_j$ should also be updated by both encoder and decoder; however, since the symbol $s_i$ was not known by those lower layers, knowledge of the symbol cannot be used to adapt their probability model contexts. Thus, the overloaded version of the update function $\texttt{update}(PM_m)\ \forall\ 0 \leq m < j$ is used which does not incorporate the symbol value in context adaptation.

As before, the rate controller design ultimately decides which symbols are coded in each layer. It therefore has a large effect on the quality of reconstruction for low bit rate decoders who

do not receive many layers. In general, a desirable trait of scalability is to code symbols that are fundamentally important to the source stream in lower layers, while allowing nonessential symbols to float to higher layers. This goal is difficult to achieve not only because it requires a semantic understanding of how important each symbol is within the stream, but also because the decision must be made based on the knowledge available at the decoder before the symbol is recovered. Furthermore, a good indicator function must also distribute enough symbols into each layer to cause a noticeable bit rate change when decoding different numbers of layers.

### 5.3.2 Partial Decoding

Suppose an encoded stream has $k$ layers in total. To reconstruct the complete source symbol stream, a decoder must receive all $k$ layered bit streams from $L_0$ to $L_{k-1}$. Alternatively, a lower quality reproduction may be recovered at a smaller bit rate by requesting only the bit streams from layers $L_0$ to $L_j$ for some $0 \leq j < k - 1$. In this case, all symbols coded in layers $j + 1$ to $k$ will not be received, but the symbols in layers $0$ to $j$ are recoverable as long as the decoder can interpret them without receiving any symbols from the higher layers. If symbols are approximately evenly distributed among the layers, then changing $j$ effectively controls a bit rate scalable trade-off between symbol recovery rate and encoded bit rate.

## 5.4 System Block Diagram

Figure 5.1 depicts a block diagram of the components of a single layer. These components include the rate controller (RC), the indicator function (I), the entropy coder (EC), the probability model (PM), and a probability model update process (U). Two streams of symbols arrive at each layer. One is a stream of symbols not encoded by any lower layer. The other is a stream of symbols encoded at a lower layer. The indicator function evaluates whether or not the next unencoded symbol should be selected for this layer. Those not chosen are passed to the next higher layer. Those chosen are passed to the entropy coder and emitted as part of this layer as well as added to the stream of symbols already appearing in a lower layer sent to the next higher layer. Symbols

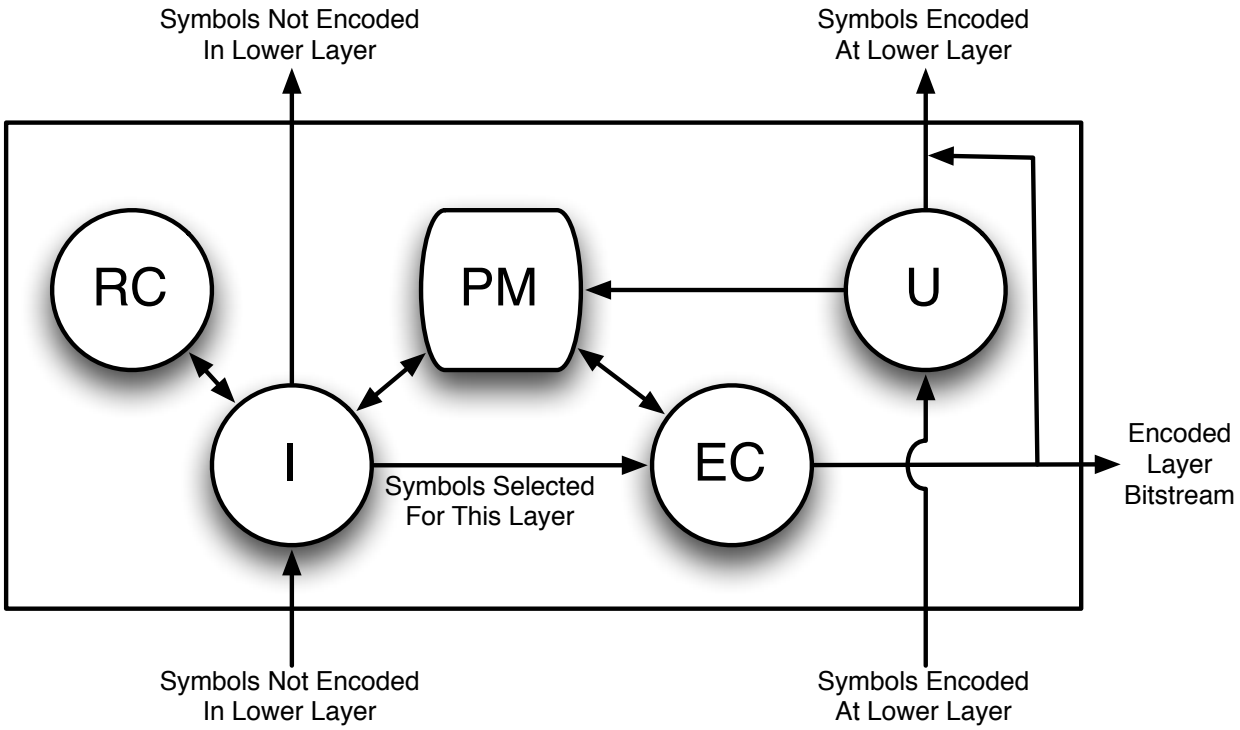



Figure 5.1: This block diagram depicts a single layer of a multilayered content-adaptive entropy encoder.

encoded as part of lower layers are used to update the probability model as if they were encoded at this layer. This allows the entropy encoding efficiency to improve at each higher layer until it is as efficient as the original representation at the highest layer.

In order for this scheme to be practical, original high-quality representations must be designed so that the symbol rate is smoothly related to reproduction quality, and sophisticated indicator functions must be developed that take into account symbol semantics—but never symbol values—to appropriately sort symbols into each layer in a rate controlled manner. In this chapter, the proposed scheme is applied to a small subset of H.265/HEVC syntax elements using a simple indicator function to demonstrate its feasibility on existing predictive coding formats. Additionally, it is applied to M-JPEG coded content to demonstrate a receiver-driven use case.

Figure 5.2 depicts the layer dependency graph for a layered, context-adaptive entropy encoder with three enhancement layers. All decoders must receive layer $L_0$, as it is the root of the dependency tree. Enhancement layer $L_j$, for $j \in \{1, 2, 3\}$, can only be decoded by a decoder that

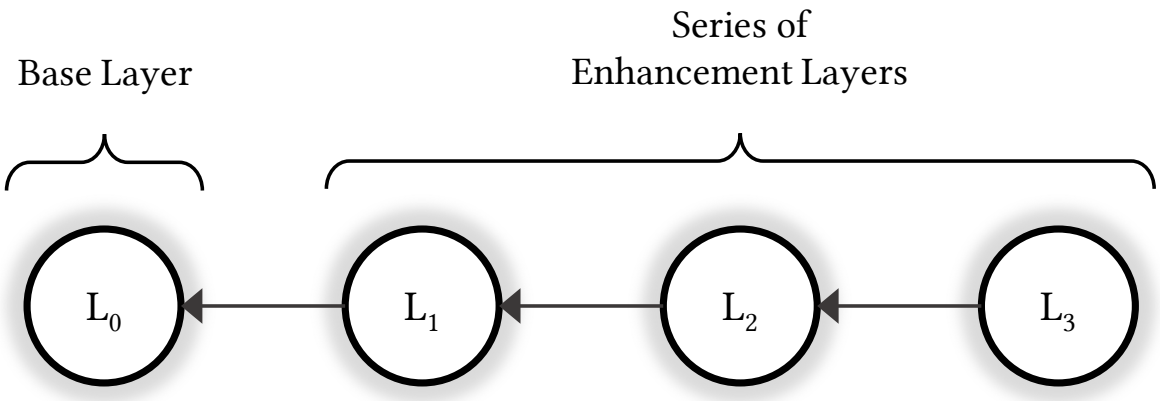



Figure 5.2: This dependency graph depicts the layers of a layered entropy coder. Each layer is dependent on the prior layer, forming a one-dimensional chain of dependent layers. No layer can be decoded without also decoding the layers on which it depends.

also has been receiving and decoding all layers below $L_j$. This is because in order to decode a symbol from layer $L_j$, the decoder must know the state of probability model $PM_j$ used to encode the symbol. But since the arithmetic coder is context-adaptive, the probabilities in $PM_j$ adapt based on previously coded symbols from layers below or equal to $L_j$. This creates a linear dependency structure between layers that maximizes coding efficiency by allowing higher layers to adapt their probability models to symbols coded in lower layers.

## 5.5 Features

This section highlights the major features of the proposed content-adaptive entropy coding algorithm, and describes how it achieves receiver-driven, top-down bit rate scalable behavior.

### 5.5.1 Top-Down Adaptation

Most existing scalable video encoding systems produce a backwards-compatible standard-compliant base layer stream along with a series of enhancement layers (Boyce et al., 2016; Wang et al., 2016). In this paradigm, the base layer stream is a complete low-quality version of the video that is fully decodable by a non-scalable decoder. To recover a higher-quality version of the video, a decoder uses one or more enhancement layers to refine and improve the base-level reproduction.

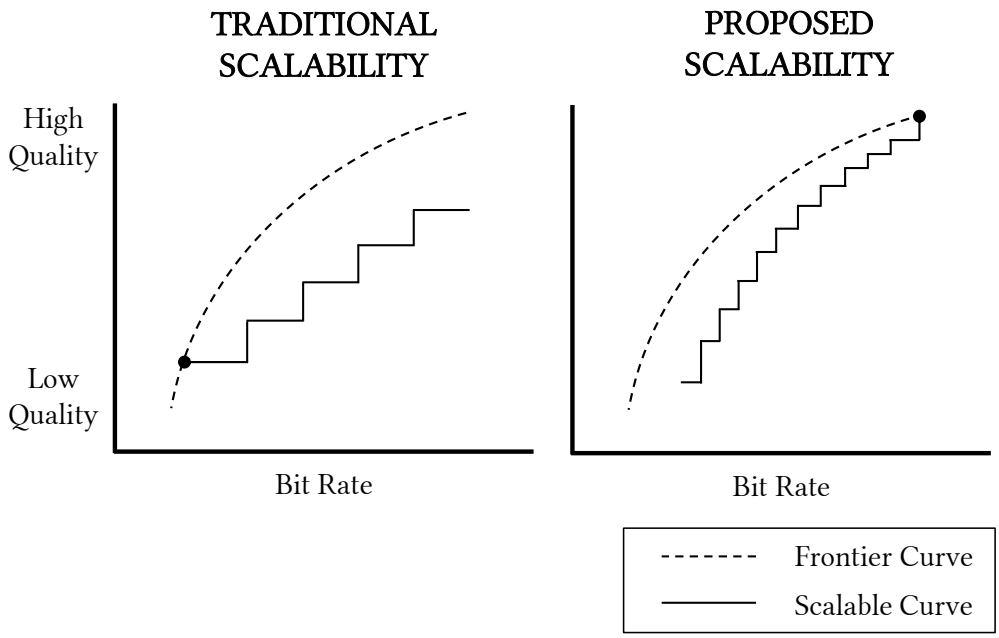



Figure 5.3: A top-down approach would provide the highest coding efficiency in requests for high-quality video.

This structure is employed by SHVC as well as its predecessor SVC (Boyce et al., 2016; Schwarz et al., 2007).

This base/enhancement layer structure induces an inverse relationship between reproduction quality and coding efficiency which is anchored at the base layer. In other words, decoding the base layer alone provides the highest coding efficiency and the lowest quality reproduction. As a decoder receives and integrates enhancement layers, the reproduction quality increases and coding efficiency decreases. Since the base layer is a fully standard-compliant non-scalable representation, it lies on the theoretically optimal rate-distortion frontier offered by the encoding algorithm. In contrast, the additional enhancement layers cannot maintain this optimal coding efficiency for higher bit rates. This is due to the fact that the presence of enhancement data when predicting future frames is not guaranteed to be available to a decoder. Thus, coding efficiency decreases as reproduction quality increases. This is illustrated in Figure 5.3. Note that the scalable curve (solid) is anchored at the base layer and intersects the frontier curve (dashed) only at this point. The area between the scalable curve and the frontier curve depicts the inefficiency incurred by introducing scalability in the scheme.

The inverse relationship between quality and efficiency illustrated above prevents practical systems from taking advantage of scalable representations. As network infrastructure continues to improve and video playback devices with increased computing capabilities become more common, we can expect high quality video delivery to be the more common use case with adaptation toward lower quality representations considered an exceptional, albeit important, condition. Existing solutions to scalability, which are most efficient for low-quality video, are therefore optimizing for a situation that is becoming less prevalent. What is required is a shift in perspective and to consider an initial high-quality representation as the starting point. That representation of the highest quality endpoint along the adaptation range is where scalability mechanisms should be anchored as close as possible to the optimal rate-distortion frontier. Adaptation to the other end of the quality spectrum can then be created as derived subsets of the information encoded at the top.

The field of scalable video coding is ripe for a fundamental paradigm shift in which scalability is no longer viewed as an enhancement process from low quality to high quality, but understood instead as an adaptive degradation process from high quality to low quality. Doing so supports myriad real-world use cases in which high-quality video is dominant and adaptation to lower-quality representations is still required. The traditional "bottom-up" structure, which is inherently optimized for low-quality reproductions (Li, 2001), is inappropriate for these use cases and must therefore be replaced.

As an exemplary alternative, the proposed content-adaptive entropy coding scheme produces a scalable encoding that takes a "top-down" approach, anchoring the rate-distortion curve at the point of highest quality and introducing coding inefficiency only at lower layers. This is illustrated in Figure 5.3. With such an approach, scalability is recharacterized not as a base layer accompanied by a series of *enhancements*, but instead as a full-quality stream separated into a series of *degradations*.

This approach has a number of benefits. First, even decoders interested in low-quality reproductions know every time a symbol is missing from the bit stream. This knowledge can be used to form a prediction about the values of the missing symbols. In particular, this trait synergizes

well with entropy coding techniques such as arithmetic coding, which require the decoder to maintain a symbol probability model. The decoder may simply query the state of the model to receive the most probable symbol(s) and use them to form a prediction about the missing value. Second, the proposed approach is domain-agnostic in terms of the source coding used as well as the type of data being coded. While we present results with video data, it could equally be used to compress voice, image, text, or any other information. Third, it is naturally structured as a top-down solution since recovering all of the layers results in receiving all of symbols of the original high-quality representation. As each layer is added, encoding efficiency improves. Finally, the approach leverages the fact that the symbol dropping decision is replicated at both the encoder and decoder to achieve scalability while keeping the decoder synchronized without requiring any additional information to be signaled.

### 5.5.2 Receiver-Driven Adaptation

In addition to providing top-down adaptation, the proposed approach also can be implemented as a fully receiver-driven system. The rate controller algorithm, which is synchronized across the encoder and decoder, makes a binary decision for each symbol about whether it should appear in the encoded bit stream. By allowing the receiver to supply this algorithm, receivers can have per-symbol control over exactly which content is transferred. At this level of control, it is easy to envision use cases where the receiver can request and receive exactly the subset of data it needs for its application. For example, consider an object tracking application. Once the region of interest has been identified, the receiver could reduce the quality of background regions in future frames in order to focus available bandwidth towards the important regions. Alternatively, consider a motion detection algorithm which could request only syntax elements associated with motion vectors from the initial encoded video. Thus, the receiver wouldn't even need to reconstruct the full video in order to know whether significant in-scene motion has occurred. The resulting system would use very little network bandwidth, and would require minimal receiver-side computation.

At the same time, for most video content, the majority of use cases will likely be homogeneous—at least in the near future. For example, users watching the video from their smartphone or a home entertainment system. For these use cases, one can imagine a system where a few well-known rate controllers are enumerated and offered by the provider. Homogeneous clients can simply select from these predefined controllers to receive their adaptation. Furthermore, to reduce provider-side computational demand, the adapted streams for the most common rate controllers can be cached and streamed on-demand from disk.

### 5.5.3 Dropped Symbol Awareness

An interesting side-effect of the proposed technique is that every decoder is aware of every symbol encoded in the initial representation—even if a particular decoder decides not to resolve certain symbols. Thus, clever decoders in the future may be designed to handle unresolved symbols differently. For instance, decoder may choose to "guess" the value of an unreceived symbol using the arithmetic coder's estimated probability model for that symbol. If the decoder's guess is correct, it was essentially signaled at zero bit cost. Decoders may guess a missing symbol value by always selecting the value modeled to have the highest probability. Alternatively, a stochastic guessing process may be implemented such that even unlikely symbols may be occasionally selected by the guesser.

### 5.5.4 Computational Efficiency

Finally, the resulting system is computationally efficient since it does not require the provider to remodel the source content as a requisite to adaptation. By treating the initial source representation as-is, the time-consuming source modeling is entirely avoided. Nonetheless, providers must be able to entropy-decode and entropy-re-encode the content, requiring minimal computation. If this requirement became a bottleneck in the long run, one could imagine a hardware implementation of the entropy encoding and decoding units used by the system that would significantly alleviate the producer computational load.

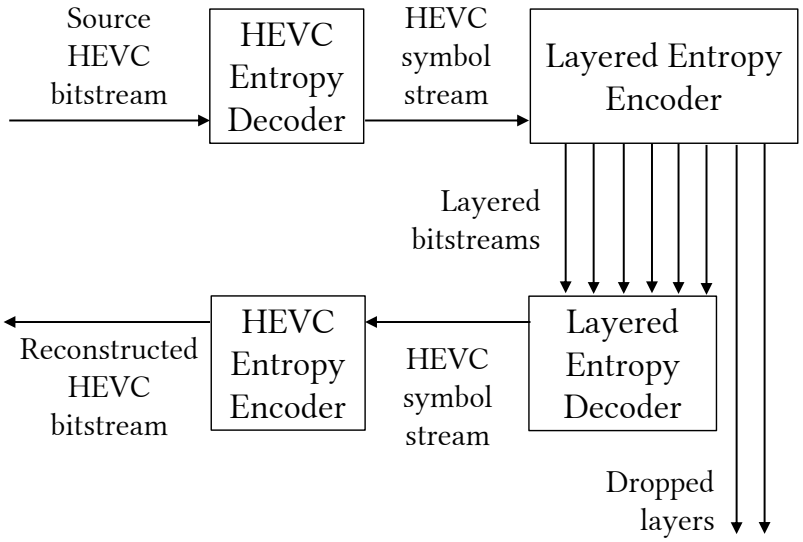



Figure 5.4: This block diagram shows the experimental setup. Note that the input and output streams are standard-compliant H.265/HEVC bit streams, while the intermediate representation is a series of entropy coded layers.

The layered interpretation of content-adaptive entropy coding also introduces additional computational modules at both the encoder and decoder for managing dependencies among layers. In particular, each layer must manage a separate probability model and bit budget state on both the encoder and the decoder. This obviously introduces memory and complexity overhead during encoding/decoding, as states must be adjusted for every layer each time a symbol is coded. The overall complexity for coding $s$ symbols into $k$ layers is therefore $O(sk)$ at both the encoder and decoder.

## 5.6 Application to H.265/HEVC

To demonstrate the feasibility of layered content-adaptive entropy coding with real data, we apply the technique to a standard-compliant H.265/HEVC symbol stream. The output adapted representation is therefore a layered series of entropy-coded bit streams, each of which provides a decoder with additional symbols from the original stream. If all symbols are received, the complete full-quality, initial representation is recovered.

As discussed in Chapter 2, H.265/HEVC is a hybrid prediction-transform codec. It is therefore organized into a source-modeling portion in which the video is represented as a stream of semantic symbols, and an entropy coding portion where CABAC and variable-length coding (VLC) serialize each symbol into the compressed output stream. Our approach is to modify H.265/HEVC's CABAC algorithm to make it output symbols into different layers according to the process described in Section 5.3. The resulting system, illustrated in Figure 5.4, acts as an H.265/HEVC transcoder which takes as input a pre-coded, standard-compliant H.265/HEVC bit stream, applies entropy decoding to recover the source-modeled symbol stream, and re-applies entropy coding via the modified CABAC to produce a series of layered bit streams. A corresponding modified decoder is also developed which receives one or more of the layered bit streams and decodes them, recovering an approximate symbol stream that can be entropy coded back into a standard-compliant HEVC bit stream.

### 5.6.1 Implementation

For this proof-of-concept experiment, only syntax elements representing residual coefficient level data are layered. The value of these elements are independent of other syntax elements and thus can be distributed among layers without having to take into account dependency relationships which are beyond the simple indicator function used (Benyaminovich et al., 2005). Furthermore, residual coefficient data makes up a large portion of the H.265/HEVC bit stream—especially when the encoder is set to output at high quality (Sole et al., 2012). Thus, layering residual coefficients is easy to implement and provides enough layerable symbols to produce a meaningful scalable encoding.

H.265/HEVC decomposes residual coefficient values into the following binary flags to be coded by CABAC:

1. `significant_coeff_flag`: Signals whether a coefficient is nonzero.
2. `coeff_abs_level_greater1_flag`: Signals whether a known nonzero coefficient is greater than one.

3. `coeff_abs_level_greater2_flag`: Signals whether a coefficient known to be greater than one is greater than two.

4. `coeff_sign_flag`: Signals a coefficient's sign using equiprobable bins.

5. `coeff_abs_level_remaining`: Signals a coefficient's remaining uncoded level using Exp-Golomb coding with an adaptive Rice parameter.

In certain scenarios, some of these flags are omitted or implicitly coded for a coefficient. In particular, H.265/HEVC limits the overall number of greater-than-one and greater than-two-flags coded in order to increase throughput (Sole et al., 2012). Furthermore, the remaining level and sign flags are always coded in bypass mode, meaning they do not use a context model for entropy coding and are directly inserted into the bit stream. See (Sole et al., 2012) for a more complete description of the H.265/HEVC coefficient coding algorithm.

Two different coefficient layering schemes are presented. In the first scheme, only bits used to code `coeff_abs_level_remaining` are allowed to be placed in higher layers. In the second, all of the following elements are layered:

- `significant_coeff_flag`
- `coeff_abs_level_greater1_flag`
- `coeff_abs_level_greater2_flag`
- `coeff_abs_level_remaining`

In the first scheme, the rate controller allots $0.25$ additional bits to each layer each time a `coeff_abs_level_remaining` value is signaled; unused allotted bits from the previous coefficient are rolled forward. Next, the bits for coding the flag via Exp-Golomb are separated and encoded one at a time into the layered bit streams, starting at the bottom layer and encoding upward. Due to the structure of Exp-Golomb codes, the decoder is able to follow this process by using the bit values decoded from one layer to know if more should be expected in the next.

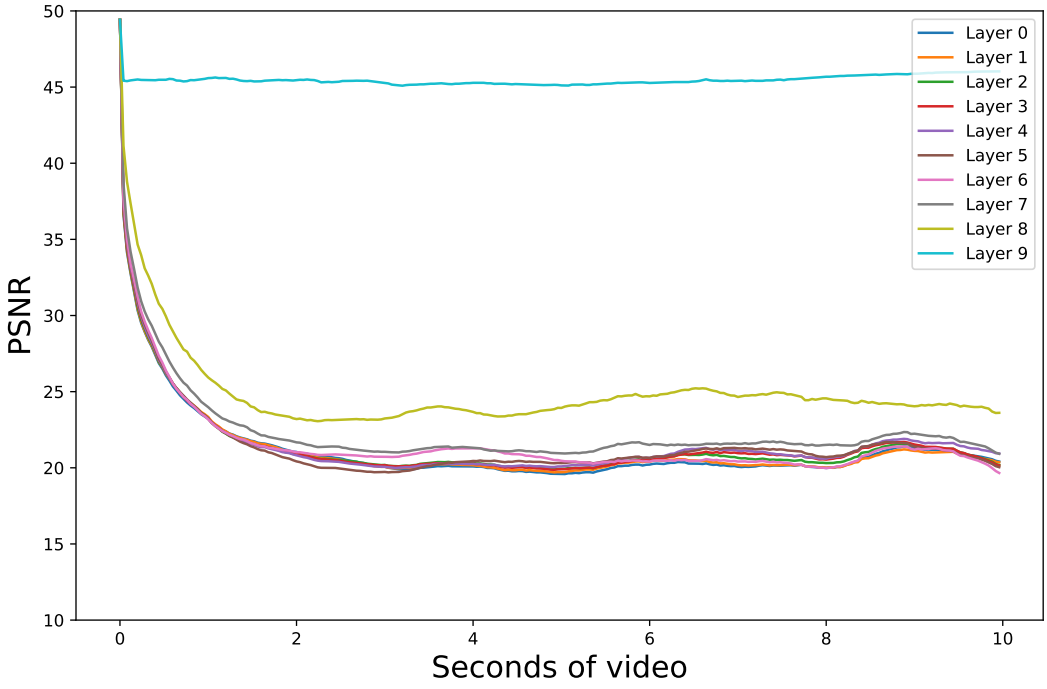



Figure 5.5: PSNR per frame when only remaining level coefficient bits are layered.

For the second scheme, all encoded individual coefficient level flags are layered. This has the added benefit of stratifying more bits into layers than the first scheme. However, it requires additional memory and complexity at both the encoder and decoder, particularly since each layer must track separate probability models for significance flags, greater than one flags, and greater than two flags. The rate controller allots $0.25$ additional bits to each layer for each *explicitly coded coefficient*. This single allotment is used to code all required flags for that coefficient. If `coeff_abs_level_remaining` must be signaled, the same process as the first scheme is used.

One concern when omitting residual data from a pre-encoded H.265/HEVC bit stream is that any error incurred by dropping symbols at the decoder was not accounted for in the DPB at encoding time; this error induces drift in the decoder's reconstructed pixel samples that can potentially be fed forward to future frames. To reduce drift, we restrict layering only to coefficient levels in inter-coded blocks, with the underlying assumption that intra-predicted blocks are more likely to be used by prediction units to code future frames. Note that this is a simplistic attempt

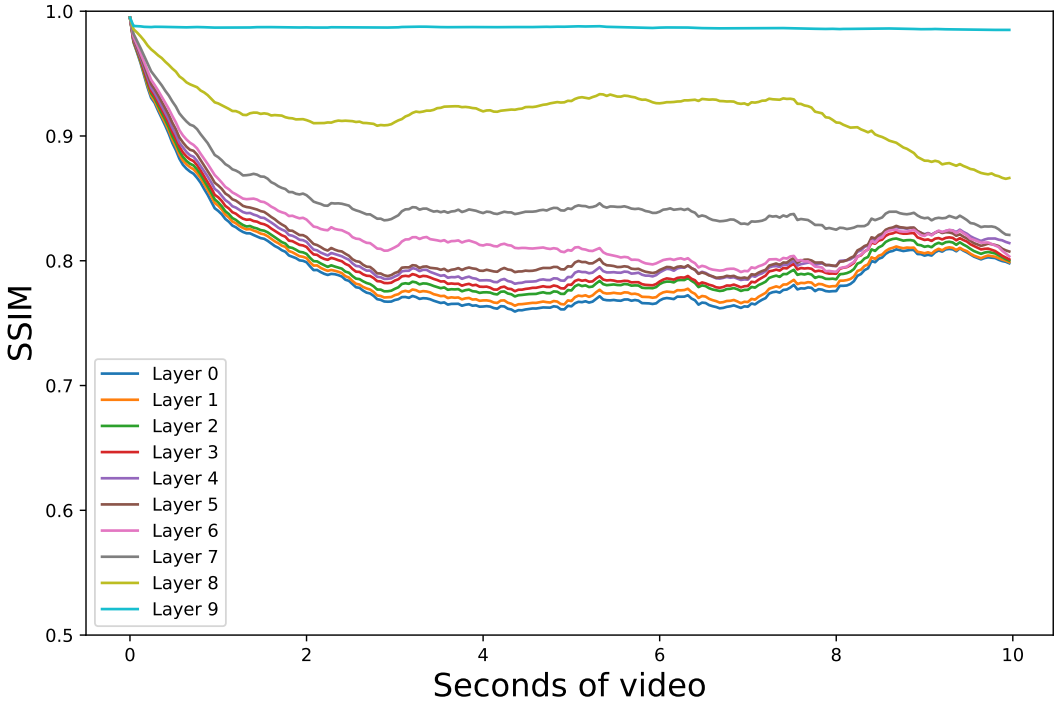



Figure 5.6: SSIM per frame when only remaining level coefficient bits are layered.

at reducing drift; we expect that more sophisticated techniques could better anticipate which coefficients are likely to be used for future prediction, and that they would achieve better results. However, it provides a valuable baseline of the feasibility of the proposed approach.

### 5.6.2 Data Set

To test the scheme, ten seconds of the 1080p, 25 fps video sequence *Tractor* is encoded using the open-source libx265 H.265/HEVC encoder (MulticoreWare, 2019) with a constant quality parameter (QP) of 17, set to produce exactly one I-frame followed by a series of P-frames. These encoding parameters are chosen to emulate a situation where a video is captured at high quality with hardware-level encoding complexity. The resulting H.265/HEVC encoding is used as the source bit stream for the two layered encoding schemes presented in this section.

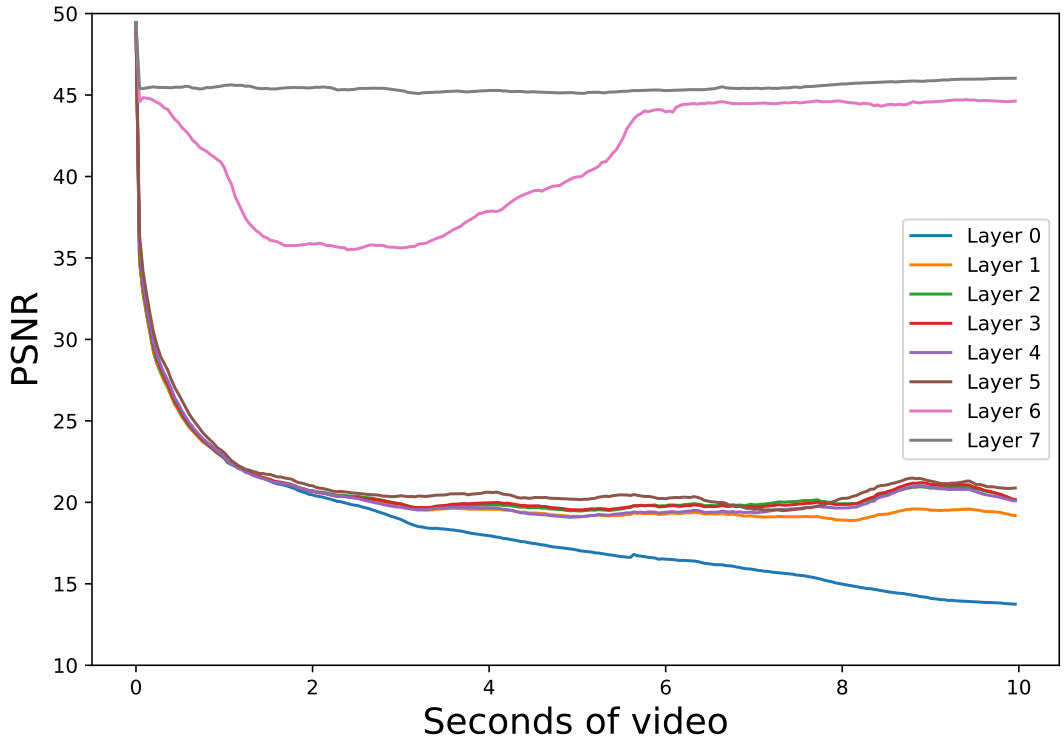



Figure 5.7: PSNR per frame when all coefficient flags are layered.

### 5.6.3 Results and Discussion

While blindly layering residual coefficient flags according to a simple fixed rate controller provides a valuable proof-of concept for top-down H.265/HEVC bit stream adaptation, the results presented here are clearly not sufficient or ready for practical use. Figures 5.9 and 5.10 demonstrate that neither of the two techniques tested were able to achieve a trade-off between quality and bit rate such that increases in rate yield corresponding increases in quality. Scores with SSIM were slightly better than PSNR, indicating that even as coefficient data is dropped, some structural information is still recoverable through the prediction and other retained elements. Moreover, the proposed technique did not yield enough layerable bits to reach the low end of the bit rate scale. Thus, future attempts at layering H.265/HEVC data must find additional syntax elements that can be removed from the bit stream without affecting decodability.

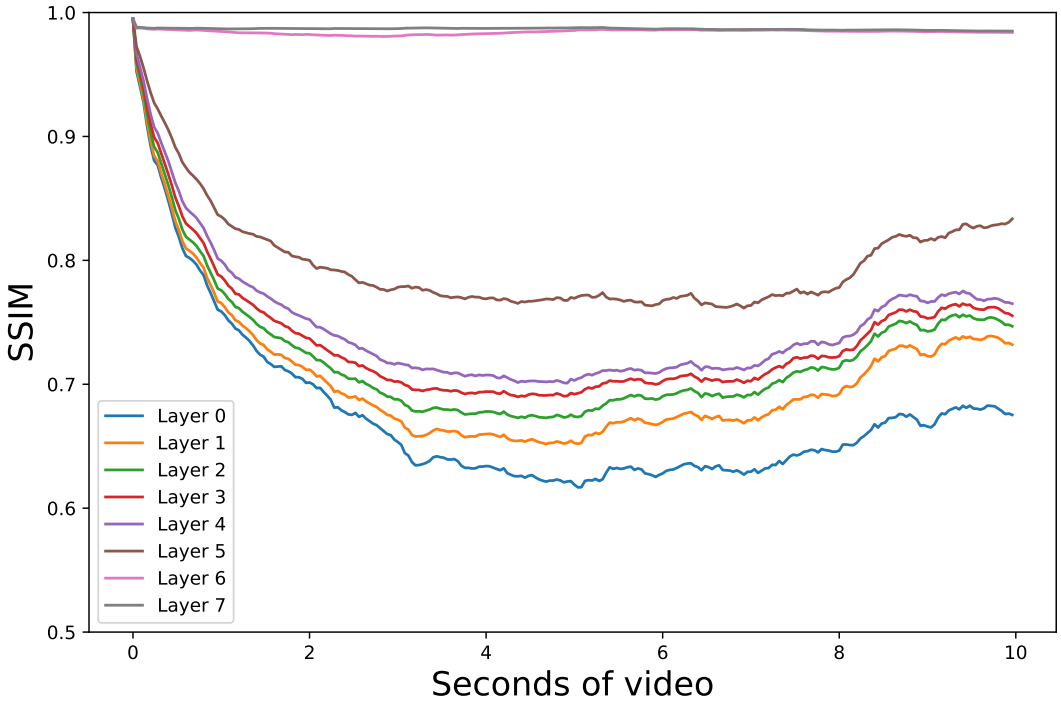



Figure 5.8: SSIM per frame when all coefficient flags are layered.

By far, the largest antagonist of both techniques is the drift caused by decoded error being fed forward by the decoder to future frames. Figures 5.5, 5.6, 5.7, and 5.8 illustrate how quickly drift takes its toll; within the first two seconds of video (roughly 50 frames), all layered streams have degraded to a near-steady state in terms of reproduction quality. Interestingly, both SSIM and PSNR demonstrate a phenomenon where quality increases slightly after a few seconds of video. This may be an indication that the probability models at lower layers require a longer period of time before they are able to adapt to the statistics of the scene and accept more flags to be encoded.

Nevertheless, the results presented here show a few promising signs that the same ideas might perform better with a more sophisticated approach. First, the rate-distortion curves for both proposed layering schemes are, as desired, anchored at the top of the frontier curve. This was the motivating concept behind the proposed scheme. Overall, the results validate that the idea can be applied to H.265/HEVC. Additionally, the equally-spaced layers shown in Figure 5.8 indicate that it

may be possible to better control coefficient removal and achieve a more gradual degradation in quality.

Finally, the results suggest a few key areas in which the scheme could be improved. First, the layered residual coefficients could be coded more carefully to reduce the chance that an important coefficient was deferred to a higher layer. Better results may be achieved by identifying these coefficients and coding them in lower layers. Second, future proposed schemes should attempt to apply layering to a larger set of syntax elements. Third, the rate controller process could be more thoughtfully designed to account for drift or symbol importance in addition to simply tracking bit rate. And finally, eliminating the effect of drift is of paramount importance. This could hypothetically be achieved by closing the prediction loop at the encoder for every layer produced, though it may make encoder complexity a concern (De Praeter et al., 2017). However, a radically different approach could be to adjust the input video's symbol stream structure to prevent long-term prediction feedback loops within the existing coding blocks. This could be accomplished by periodically signaling small portions of the video without using prediction such that eventually the decoder's entire picture is refreshed.

Overall, this work demonstrates a baseline approach towards entropy-driven, top-down scalable video coding. Although the results achieved are not production-ready, they are important because they represent a seminal point in the right direction for the future of video adaptation and have the potential to spur this change. In particular, these results demonstrate that a top-down interpretation of video coding is not only possible, but it is a fundamentally more representative interpretation of modern video usage behavior.

## 5.7 Application to M-JPEG

The prior section described an initial implementation of content-adaptive layered entropy coding as applied to an input H.265/HEVC stream. The results show that top-down adaptive encoding can be created with modern codecs. In particular, the rate-distortion curve produced was successfully anchored at the top. However, the trade-off between rate and distortion induced by

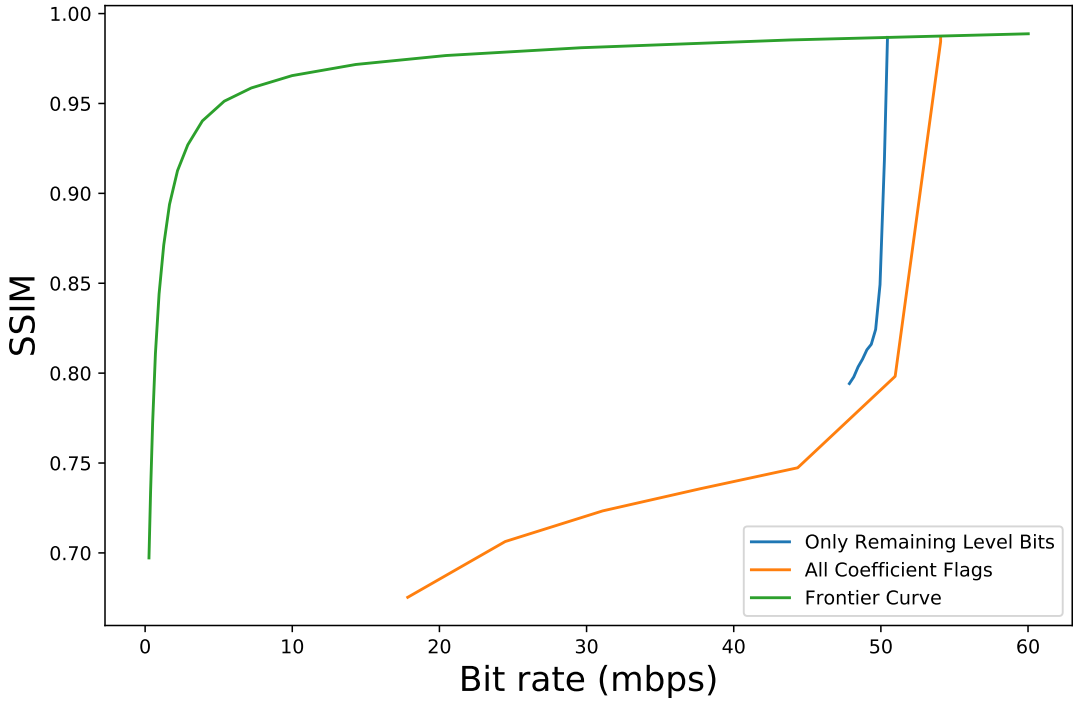



Figure 5.9: Rate-distortion curve showing the two proposed coefficient-dropping techniques in comparison to the frontier using SSIM as the quality metric.

the coded layers is not sufficient for practical use. Analysis of this initial experiment indicates that the challenges faced by top-down adaptation of H.265/HEVC coded bit streams is due to the inherent design of the codec. We identify two design decisions as particularly problematic. First, most syntax elements of an H.265/HEVC bit stream are tightly dependent on each other. This forces most syntax elements to be pushed to the base layer, and greatly limits the syntax elements that are actually available to be layered. Second, H.265/HEVC is a highly predictive codec. This is generally a favorable trait because it results in high coding efficiency, but in this case means that any missing syntax elements at the decoder result in decoder drift that is quickly amplified.

These results do not mean that top-down adaptive coding is always ill-suited to predictive coded data at large. We cite as future work the development of a novel predictive video codec that is better suited to the content-adaptive entropy coding approach presented by this chapter. The

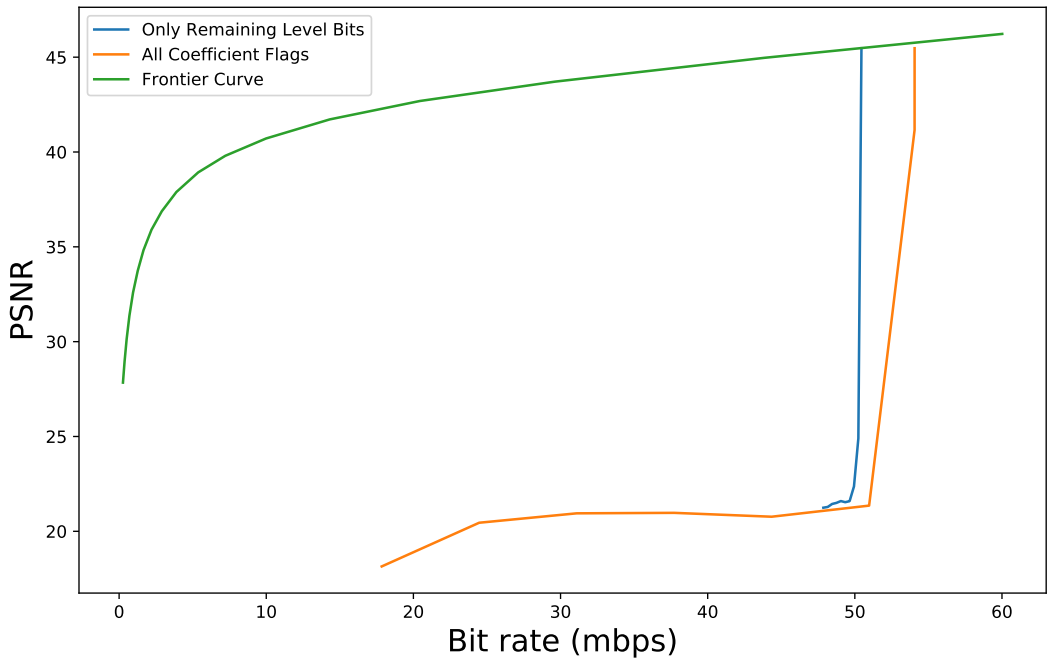



Figure 5.10: Rate-distortion curve showing the two proposed coefficient-dropping techniques in comparison to the frontier using PSNR as the quality metric.

results do indicate, however, that a different strategy is needed for adapting H.265/HEVC data in a top-down manner. Such an alternate strategy is proposed in Chapter 6.

Nonetheless, context-adaptive entropy coding can still be successfully implemented to provide top-down, receiver-driven adaptation of data encoded with existing video formats. The key is to adapt video content that was originally captured using an initial encoding that is not highly predictive and does not contain primarily inter-dependent symbols. M-JPEG is one such existing codec.

This section presents another implementation of context-adaptive entropy coding. This time, the technique is applied to video content encoded with M-JPEG. The results demonstrate that the proposed adaptive approach works well when applied to a non-predictive, non-dependent symbol stream. Furthermore, by allowing smart receivers to drive the adaptation process, we show that the available network bandwidth can be better targeted to the receiver's application. Ultimately, this allows our approach to beat an equivalent non-scalable encoding in specific use cases.

### 5.7.1 Codec Selection

The main issues with the previous experiment that limited the results were that video data encoded with H.265/HEVC is highly predictive and highly inter-dependent. Thus, this experiment set out to demonstrate the application of top-down receiver-driven content-adaptive entropy coding on a non-predictive codec with independent syntax elements. M-JPEG is a natural choice, both for these qualities and for its widespread use as an initial coding format for video content produced by high-end consumer and professional-grade digital cameras.

Ultimately, video content coded with M-JPEG is simply a sequence of independently-coded JPEG images, each representing a separate frame of video data. No attempt is made to apply prediction of any kind, including inter-frame and intra-frame prediction. This makes M-JPEG a great choice for top-down adaptation.

M-JPEG pixel values are encoded by first partitioning the given frame into minimum coded units (MCUs) of 8x8 or 16x16 pixels each. Each MCU is then transformed using the implementation of the DCT described in Chapter 2. Next, the resulting frequency coefficients are quantized using a predefined quantization table to apply loss in the frequency domain. They are then serialized using zig-zag scanning, which proceeds through all quantized coefficients starting from the top-left of the block and scanning through to the bottom-right. Finally, the scanning process yields a stream of integers which is encoded to a bit stream with entropy coding techniques. M-JPEG supports both Huffman entropy coding and arithmetic coding; however, due to historical reasons, almost all prevalent JPEG codec implementations (including the ones built-in to modern web browsers) only support reading and writing JPEGs that are coded with Huffman entropy coding. That said, the two techniques are equivalent, and any JPEG content that uses one entropy coding method can easily be losslessly converted to the other.

### 5.7.2 Implementation

The JPEG standard optionally allows for an arithmetic coding unit to be used for entropy coding as a replacement of Huffman coding. Since this experiment is intended to evaluate content-

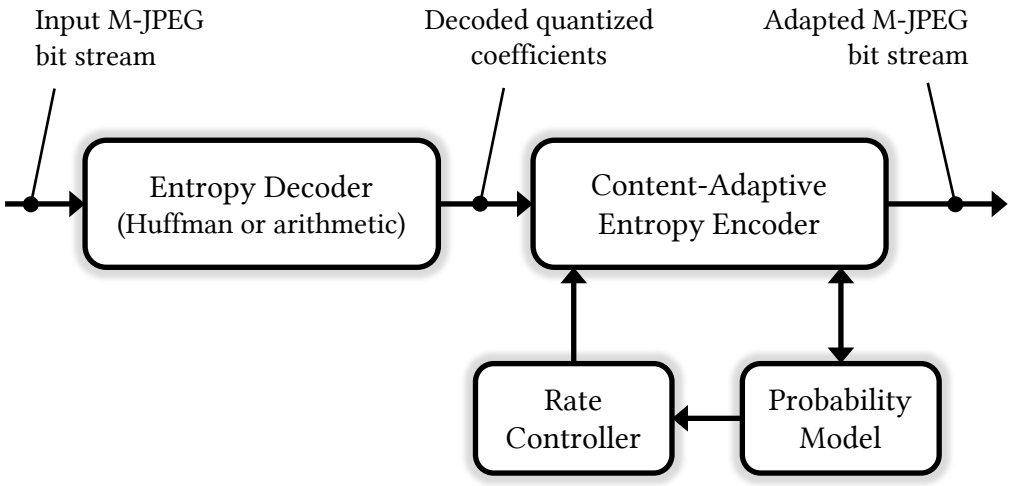



Figure 5.11: This block diagram depicts the M-JPEG adaptive encoder architecture. The system entropy-decodes an input M-JPEG bit stream, and applies content-adaptive entropy encoding to the decoded quantized coefficients using an application-specific rate controller.

adaptive entropy coding, which is essentially a modification to arithmetic coding, it makes sense to use the JPEG specification's embodiment of arithmetic coding for adaptation. Thus, a modified M-JPEG encoder and corresponding decoder has been developed where the entropy coding units have been replaced by a modified version of JPEG's arithmetic coding algorithm that implements content-adaptive entropy coding. The encoder and decoder system architectures are depicted in Figures 5.11 and 5.12, respectively. The remainder of this section details the design aspects of the modified codec.

M-JPEG codes each video frame sequentially in display order. Each MCU of a given video frame is independently coded, one at a time, in raster order. To code an MCU, the quantized coefficients that constitute it are entropy coded, one at a time, in zig-zag order starting from the top-left and ending at the bottom-right of the block. It is during this coefficient entropy coding process that the content-adaptive modification is made.

According to the M-JPEG specification, each quantized coefficient is entropy coded as a series of binary decisions which are ultimately serialized to the bit stream with a CABAC. The quantized coefficient's sign is coded as a binary decision. Next, its "category" is coded, which is essentially the position of the first nonzero bit. Finally, the coefficient's offset within its category

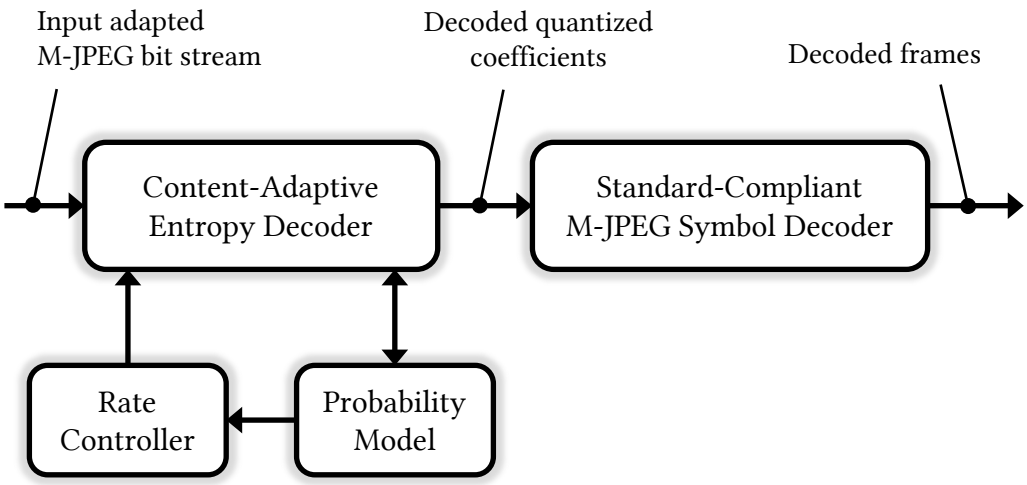



Figure 5.12: This block diagram depicts the M-JPEG adaptive decoder architecture. The system decodes an adapted input M-JPEG bit stream using an application-specific rate controller.

is coded. Technically, each of these decisions are independent symbols with respect to entropy coding. However, for the purpose of this implementation, the entire quantized coefficient is treated as an atomic symbol. In other words, adaptation decisions are made on a per-coefficient basis: each quantized coefficient is always either fully encoded or fully dropped, based on the rate controller's decision. Note that a different implementation could allow for partial coefficients to be coded by dropping some of the binary decisions that constitute them. However, other than giving even finer-grained control to the rate controller, the benefits of such an approach on rate-distortion performance are likely insignificant in the vast majority of use cases.

The rate controller algorithm used for this implementation of content-adaptive entropy coding is the simple bit-budgeted controller described in Section 5.2. For each quantized coefficient (i.e. for each encoded symbol), an appropriate value ultimately must be selected for $b$, the maximum number of bits allocated to encoding the symbol. The value of $b$ must be carefully chosen to achieve an overall target bit rate for the video, but can also take into account local context information and receiver-driven directives. The value of $b$ for a given coefficient is determined as follows. First, the receiver specifies an overall target video bit rate, which is distributed among the MCUs for each video frame. By default, each MCU of each frame receives the same rate allocation for coding

its coefficients. However, the receiver has the option to specify a different distribution paradigm. Smart receivers can use this functionality to direct bit allocation towards different regions of interest within the frame. Each MCU therefore receives its own allocation of available bits (i.e. its "bit budget"), and is responsible for expending those bits to code coefficients.

The array of quantized coefficient-symbols that constitute an MCU begins with a DC frequency coefficient followed by a variable-length sequence of AC frequency coefficients. The bit budget $b$ is always set to infinity for the DC coefficient. That way, the DC coefficient is always encoded (i.e. $\delta(\infty, PM) = 1$ is always true). Next, provided that the MCU has not yet expended its bit budget, AC frequency coefficients are encoded one at a time until no more bits remain. In other words, the $b$ value used to encode each AC coefficient is simply the remaining bit budget for the MCU. Once the bit budget runs out, encoding of the MCU has finished, and the algorithm proceeds to the next MCU. If all AC coefficients are successfully encoded before the bit budget runs out, any surplus bits are rolled over to the next MCU.

In addition to specifying the overall target bit rate for adaption, receivers may also provide a strategy for allocating bits to coefficients. In other words, they can indicate the value of $b$ for each coefficient. By giving receivers full control over symbol inclusion, the resulting adaptation is truly receiver-driven. We anticipate that in the future, smart clients will be able to use this flexibility by developing sophisticated strategies for symbol selection that are tightly integrated with their applications. For example, receivers interested in a particular region of interest (ROI) within the scene can control adaptation through the presented architecture so that MCUs within the ROI are prioritized.

In fact, this idea of ROI-directed, receiver-driven adaptation has been implemented and evaluated using the proposed architecture. The objective of this experiment is to present a receiver-driven use case for the proposed adaptation technique, and demonstrate how it can outperform a non-receiver-driven alternative for special use cases. Therefore, three different smart receivers are implemented and compared when requesting the same video content. For the baseline case, one receiver simply requests the content at a low bit rate such that available bits should be allocated

| | *Tractor* | *Johnny* | *Park Joy* |
|---|---|---|---|
| **Resolution** | 1920 x 1080 | 1280 x 720 | 1920 x 1080 |
| **Color Depth** | 24 bits | 24 bits | 24 bits |
| **Frame Count** | 761 | 600 | 500 |

Table 5.1: Descriptive statistics of content used for M-JPEG content-adaptation experiment

evenly across the scene. In this case, no receiver-directed adaptation is used. A second receiver is particularly interested in a specific region of the frame, and specifies the region with a bounding box. That receiver therefore receives an adapted version of the content where pixels in the ROI are given higher bit allocation, at the expense of background pixel accuracy. Finally, a third receiver has a very specific understanding of the content it needs. It therefore requests that the available bit budget be expended primarily within a small, free-form ROI that it specifies as a compressed bit mask as part of its request. These three receivers are purposefully designed to demonstrate that high specificity in a receiver's request can yield higher compression at little or no loss of quality within regions of interest.

### 5.7.3 Data Set

To demonstrate the proposed technique on a variety of content, three uncompressed public-domain videos were selected to be source material. In all three videos, a clear subject area exists that could feasibly be identified by a smart receiver. Sample frames from each video are reproduced in Figure 5.13, and descriptive statistics about each are presented in Table 5.1.

### 5.7.4 Results and Discussion

The results presented in this section illustrate the performance of content-adaptive, receiver-driven entropy coding when applied to M-JPEG data. Three different video content sources were tested: *Johnny*, *Tractor*, and *Park Joy* (see Figure 5.13 and Table 5.1). Each video is initially encoded using M-JPEG, and then is requested by three different types of simulated receiver.

Figure 5.13: Three videos depicting different scenes types are used in the M-JPEG content adaptation experiment. Sample frames from each are reproduced above. From top: *Johnny*, *Tractor*, *Park Joy*.

1. The *Baseline* receiver requests adapted content at a specific target bit rate without specifying any particular region of interest. The video data adapted for the *Baseline* receiver therefore has a relatively homogeneous bit rate allocated to each frame and to each MCU within a frame.

2. The *Bounding Box* receiver emulates a smart client that is actively performing object tracking on the video content. Since the *Bounding Box* receiver is specifically concerned with its self-identified region of interest, it is able to specify to the provider the bounding box for the region it wants. The video data adapted in response therefore targets this region of interest, and purposefully allocates more bits to that region.

3. Finally, the *Contour* receiver emulates a smart client that knows exactly which MCUs it cares about, and specifies this information to the provider through a bit mask embedded in its request. Thus, the adapted video content it receives can be targeted even more to the precise areas of interest.

Figure 5.14 depicts selected frames received by the *Baseline* receiver. Two frames are presented from each sample video to illustrate the difference between the highest and lowest quality video received by the *Baseline* receiver. These represent the extremes of adaptation offered by the proposed technique.

Figure 5.15 shows frames received by the *Bounding Box* receiver. The frames in the first column provide a visualization of the bounding boxes used by the receiver to specify its ROI. The second column contains frames from the lowest possible quality videos that the *Bounding Box* receiver could potentially request.

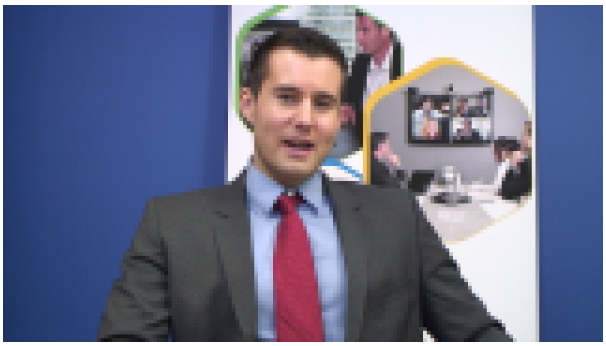
Figure 5.16 is similar to Figure 5.15, except it depicts frames received by the *Contour* receiver. This receiver provides its ROI as a bitmap instead of as a bounding box. Therefore, the ROI is able to have much more detail. Sample masks supplied by the *Contour* receiver are shown in Figure 5.17. The bit masks provided by the receiver are one-eighth the resolution of the source

video, since only one bit must be specified for each $8 \times 8$ MCU, indicating whether the MCU is part of the region of interest.

Figure 5.18 illustrates the rate-distortion curves achieved by the proposed method for each content type and receiver. Full-frame PSNR is the quality metric used for these figures, even though *Bounding Box* and *Contour* receivers direct their available bit rate towards the receiver-supplied ROIs (as shown in Figures 5.15 and 5.16). Overall, the trade-off between bit rate and quality offered by this approach is quite good. With end-to-end compression that includes a source modeling component, one would normally hope to see a flat rate-distortion curve that curves downwards only at the lower bit rates. However, the content being adapted for this experiment has already been source-modeled. If symbols could simply be eliminated from a pre-encoded M-JPEG stream with little to no quality loss, then those symbols would likely already have been removed by the specification. Therefore, for a pre-coded M-JPEG stream, we expect each symbol dropped to have an effect on the recovered quality. Indeed, this behavior is confirmed by the linear rate-distortion curves depicted.

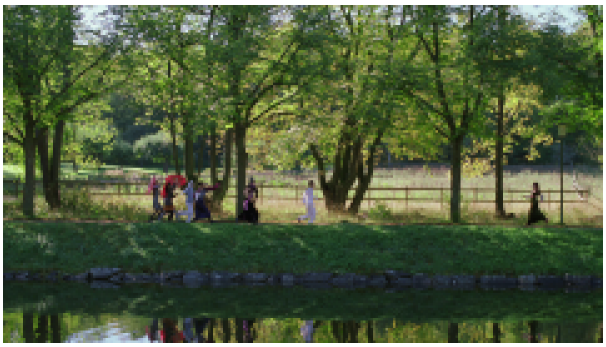
Finally, Figure 5.19 shows the bit rate required for each receiver to obtain the primary content of each video at full quality. Since the *Baseline* receiver does not direct the adaptation process, a full-quality version of the primary content can only be obtained if the receiver requests full-quality content everywhere. In contrast, the *Bounding Box* and *Contour* receivers are able to receive their specified ROIs at full-quality at much lower overall bit rates. This is because they direct the adaptation process according to their application needs. Furthermore, the *Contour* receiver outperforms the *Bounding Box* receiver, due to the additional specificity afforded by the bitmask-based ROI. Overall, this result should be interpreted as evidence that receiver-driven adaptation is able to outperform a non-receiver driven approach for targeted use cases.

(a) Highest bit rate, *Johnny*



(b) Lowest bit rate, *Johnny*

(c) Highest bit rate, *Park Joy*



(d) Lowest bit rate, *Park Joy*

(e) Highest bit rate, *Tractor*

(f) Lowest bit rate, *Tractor*

Figure 5.14: This figure compares high-bit-rate and low-bit-rate frames extracted from video content received by the *Baseline* receiver.

(a) Visualization of bounding box overlay, *Johnny*

(b) Lowest bit rate, *Johnny*

(c) Visualization of bounding box overlay, *Tractor*

(d) Lowest bit rate, *Tractor*

(e) Visualization of bounding box overlay, *Park Joy*

(f) Lowest bit rate, *Park Joy*

Figure 5.15: This figure illustrates sample frames received by the *Bounding Box* receiver, as well as an overlay depicting the requested region of interest.

(a) Visualization of contour overlay, *Johnny*

(b) Lowest bit rate, *Johnny*

(c) Visualization of contour overlay, *Tractor*

(d) Lowest bit rate, *Tractor*

(e) Visualization of contour overlay, *Park Joy*

(f) Lowest bit rate, *Park Joy*

Figure 5.16: This figure illustrates sample frames received by the *Contour* receiver, as well as an overlay depicting the requested region of interest.

(a) Visualization of contour overlay, *Johnny*

(b) Contour mask, *Johnny*

(c) Visualization of contour overlay, *Tractor*

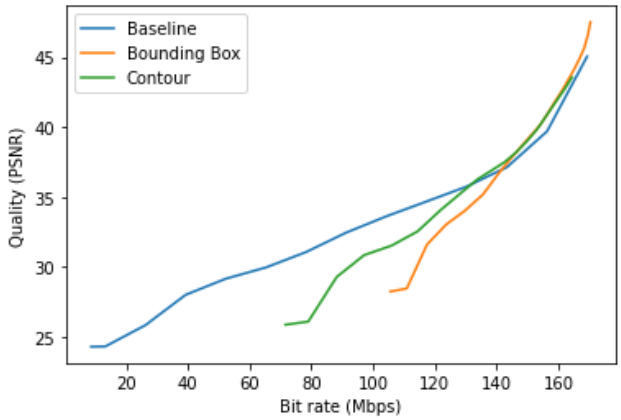
(d) Contour mask, *Tractor*

(e) Visualization of contour overlay, *Park Joy*

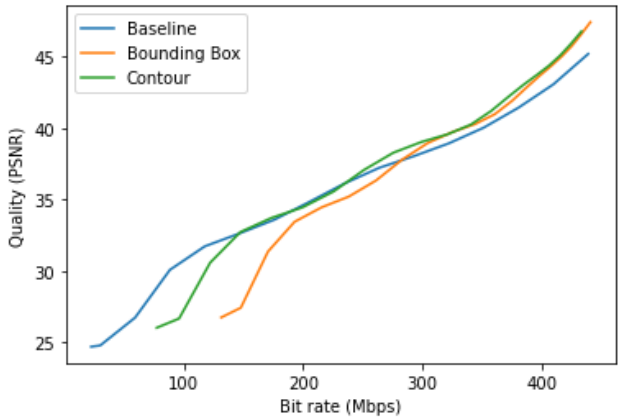
(f) Contour mask, *Park Joy*

Figure 5.17: This figure illustrates sample masks supplied by the *Contour* receiver.

(a) Full-frame rate-distortion performance for all three receivers requesting *Johnny* content.




(b) Full-frame rate-distortion performance for all three receivers requesting *Tractor* content.




(c) Full-frame rate-distortion performance for all three receivers requesting *Park Joy* content.

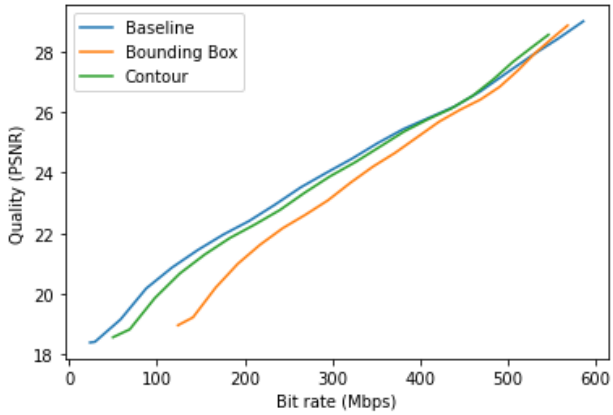



Figure 5.18: This figure shows the rate-distortion performance of adapting M-JPEG content. For *Bounding Box* and *Contour* receivers, pixels in the region of interest are prioritized by the rate controller over those in the background, and therefore are transmitted at high quality.

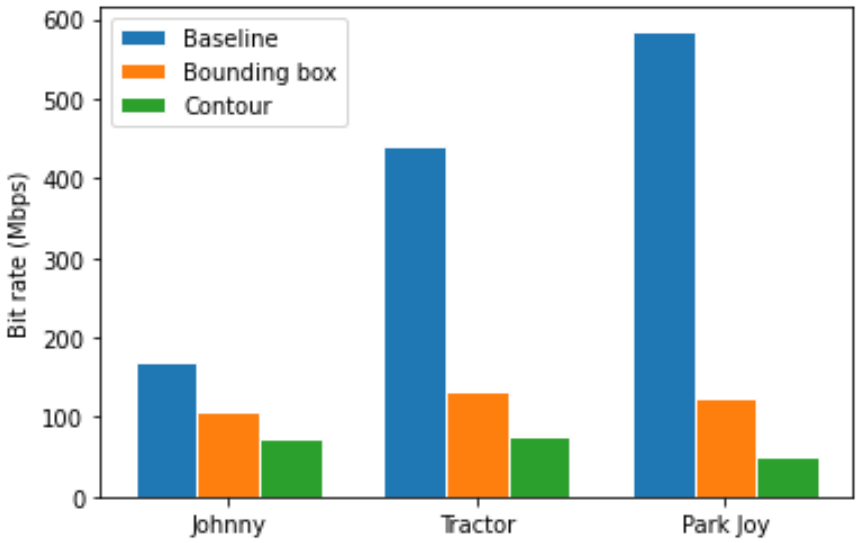



Figure 5.19: For each receiver and content type, this figure depicts the bit rate necessary to transmit the region of interest at full quality using content-adaptive M-JPEG.

## 5.8 Takeaways

This chapter presented a fundamental, entropy-based adaptation approach that is receiver-driven and prioritizes the initial encoding as the ground truth. A major strength of this approach is its generalizability: since it operates on an abstract symbol stream, it can be applied to any content source to achieve receiver-driven adaptation. And since it operates directly on the initial encoded symbols, it provides top-down adaptation for clients who have the available bandwidth to receive the full stream. Finally, by avoiding recomputation of the content's source modeling, it minimizes provider-side computation.

The application of content-adaptive entropy coding to H.265/HEVC coded data showed that modern predictive video codecs are not well-suited to top-down adaptation as currently defined. This does not mean that predictive coding itself is fundamentally opposed to top-down adaptation. However, it does suggest that there is room for development of new predictive codecs for video that provide a more flexible representation of their content. This is a major insight afforded by the work presented in this chapter. Development of a new video codec that can compete with the

state-of-the-art stands to be a major, time consuming challenge. We therefore leave this task to future work.

Finally, the series of M-JPEG experiments presented in this chapter demonstrate a successful application of content-adaptive entropy coding. By allowing receivers to drive the process of adaptation, they are able to tailor the data they receive according to their diverse needs. A myriad of future work could be devoted to applications of this technology. One idea is an application of multiple description coding (MDC), where different receivers request the same video content with slightly different rate controllers. If each rate controller produces an adapted stream that prioritizes different subsets of the source stream, these disparate adapted streams could be combined together to recover a higher version of the original.

Ultimately, this work represents the initial steps towards development of myriad new applications for content-adaptive entropy coding. We look forward to the realization of many of these ideas as future work.

## 5.9 Future Work

This chapter introduced a novel, fundamental approach for content-agnostic adaptation that works in the entropy layer. To demonstrate its merit, the technique is applied to video data encoded with two well-known, commercial video codecs: H.265/HEVC and M-JPEG. These experiments evidence how symbol dropping is a compelling approach for adaptation, both for its flexibility to work with myriad content types and for the computational elegance of adapting without re-modeling the source content. The research presented by this dissertation is best understood as the seminal, foundational work toward entropy-based content adaptation. In that respect, much work is still needed to explore how this strategy interacts with real-world data before it is ready to be applied to production environments. The remainder of this chapter provides a "roadmap" of future work, detailing some of the most immediate tasks that must be completed in pursuit of wider adoption of entropy-based content adaptation techniques for real-world data.

### 5.9.1 Drift Management for Predictive Codecs

Decoder drift was a major factor in the H.265/HEVC symbol dropping experiment. While we anticipated that some drift would occur—especially for the lowest bandwidth receivers—the significance of drift on the recovered signal exceeded expectations. Nonetheless, highly predictive codecs are rapidly gaining market share at the time of publication of this dissertation. All new codecs, including H.265/HEVC, H.266/VVC, VP9, and AV1, aim to achieve higher compression rates for high-bandwidth video primarily by increasing the sophistication of prediction. Therefore, in the near and medium-term future, entropy-based content adaptation will only be widely applicable to pre-encoded video data if methods can be developed to better support these highly predictive codecs. Since drift is the major obstacle, research is needed for ways to avoid, mitigate, or compensate drift that occurs from symbol dropping.

Below, three possible approaches for mitigating the drift problem are presented as future work opportunities. Frame-based drift avoidance would eliminate drift by only applying symbol dropping to frames that will not be used for reference in future prediction. Drift tracking would incorporate a pixel-space decoder which tracks the regions of a frame where drift has occurred. Finally, statistical block-wise IDRs would stochastically rewrite regions of pixels so that non-predictive coding is used, thereby eliminating any drift that may have built up in those regions.

#### Frame-Based Drift Avoidance

Fundamentally, decoder drift only occurs when—unbeknownst to the encoder—an imperfectly reconstructed frame is used as reference by the decoder to predict a future frame. Therefore, every symbol dropped by the entropy-based rate controller which should have helped form a frame that is later used for prediction contributes to decoder drift. One possible strategy for eliminating decoder drift is therefore to target symbol dropping only towards frames which will not be used for reference in later prediction. Modern video codecs classify encoded frames as I-frames, P-frames, and B-frames, based on how they are used for prediction (see Chapter 2). In H.265/HEVC, I-frames and P-frames are eligible to be used in future prediction; B-frames are not eligible. Therefore,

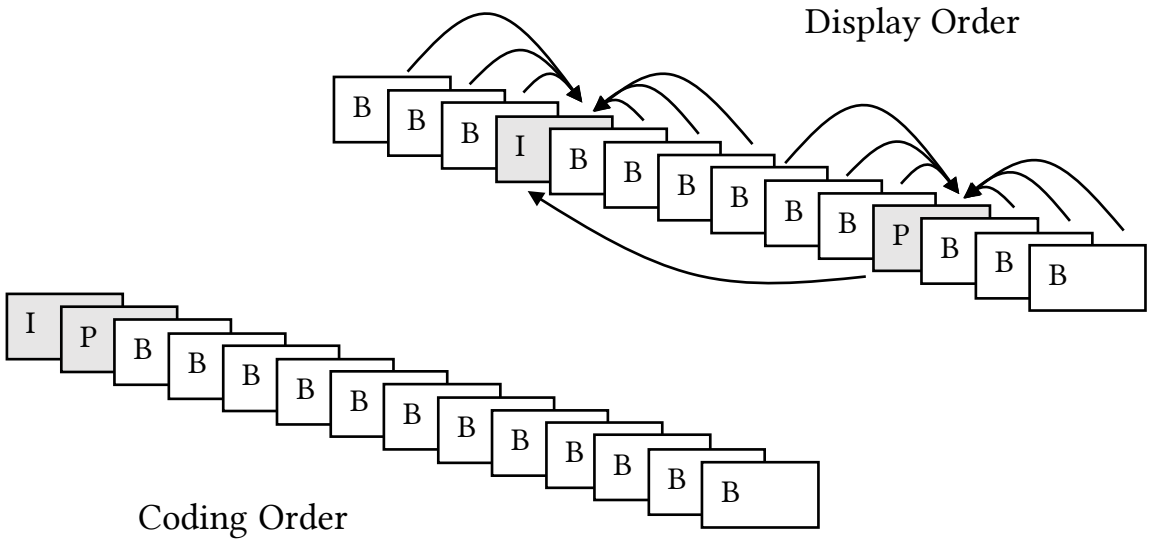



Figure 5.20: A sequence of video frames for layered entropy coding are depicted in coding order and display order. In both orders, frames to the left are coded or displayed earlier than frames to the right. The I-frame and P-frame are coded in the base layer, ensuring that all decoders will have them available for prediction. The B-frames are predicted using the temporally nearest of these two frames, and the produced syntax elements are layered. To prevent decoder drift, the layered B-frames are never used for future prediction. The arrows associated with the display ordering indicate coding dependencies between frames.

temporal-based (i.e. motion compensated) decoder drift could be eliminated by simply restricting symbol dropping only to the symbols that signal a B-frame.

Targeting symbol dropping only to B-frames will not eliminate drift entirely. In particular, any intra-frame (i.e. texture-based) prediction used to encode the B-frames will still incur decoder drift. For example, suppose the rate controller drops symbols while adapting a coding unit which is part of a B-frame. Thus, the decoder's pixel-domain reconstruction of that coding unit will be imperfect, reflecting the missing symbols. Now, suppose a neighboring adjacent coding unit within the same B-frame is formulated using intra-frame prediction. If the coding unit specifies an intra-coding mode that causes its prediction to be extrapolated from the imperfectly reconstructed neighboring block, then the pixel value errors from that block will be carried forward to the next one. This is decoder drift. However, since B-frames are guaranteed to never be referenced for

future prediction, any drift which occurs during signaling of these frames will never carry forward temporally.

Given this analysis, the proposed approach can be expected to result in an adapted version of the video in which all I-frames and P-frames are perfectly reconstructed at the decoder, and only B-frames are adapted to hit the target bit rate. Depending on how many symbols are dropped from the B-frames, this will likely result in a noticeable difference in perceived quality between decoded I-frames, P-frames, and B-frames. Moreover, another challenge is that B-frames are typically easiest for an encoder to signal, since they have the most reference content available at the decoder to use for prediction. In other words, they generally represent the smallest proportion of the overall encoded bit stream, when compared with I-frames and P-frames. Thus, limiting symbol dropping only to B-frames would greatly limit the percentage of coded symbols which could be dropped feasibly to achieve adaptation.

These issues—namely, that only B-frames will exhibit a perceived decrease in reconstructed quality, and that the number of symbols eligible to be dropped will be small—can be mitigated if the original H.265/HEVC encoder can be controlled. For instance, suppose the initial encoding is organized in a way such that a single I-frame is followed by a single P-frame, and then both are followed by a long stream of B-frames which all predict back to the initial I-frame and P-frame. In such a scenario, most of the encoded frames are B-frames, which would be eligible for symbol dropping without incurring inter-frame decoder drift. Thus, perceived quality on a frame-by-frame basis would appear to be roughly even across each sequence of B frames. At the same time, more symbols overall would be eligible to be dropped. However, as playback time passes, later frames in the B-frame sequence would have a harder time predicting from the old I-frame and P-frame, due to large temporal displacement. Some level of coding efficiency loss would therefore occur, especially for later B-frames in a sequence. Eventually, a new I-frame and P-frame would need to be signaled to provide new content that can be used as the basis for prediction. Ultimately, such an approach could fix the limitations described above, but is only an option if the initial encoder can be tuned

appropriately to produce a standard-compliant H.265/HEVC stream where frames are coded using the necessary I-P-B-B-B-B-B sequence.

### Drift Tracking

Instead of carefully avoiding decoder drift by further limiting the subset of layered symbols, another approach is to accept the drift and deploy a mitigation strategy to minimize its effect in the long term. Thus, one idea is to layer the residual coefficient symbols as described earlier in this chapter for the original H.265/HEVC experiment. However, each time a symbol is dropped, the layered encoder marks the resulting pixel block as "dirty," indicating that the corresponding partial decoder has an imperfect reconstruction of those pixels. Various layers of sophistication could be used here; the simplest approach would be to simply incorporate a bit field for each encoded layer and each frame. Anytime a symbol is dropped, the corresponding block would be marked as dirty. Similarly, anytime a dirty block of pixels are referenced for prediction, the predicted block must also be marked as dirty. This would allow for a decoder to track which blocks of pixels have experienced decoder drift.

A more sophisticated approach is for the layered encoder to track the precise amount of drift experienced by a decoder at each layer. This requires the layered encoder to maintain a separate DPB for each decoded layer, representing the imperfect reconstructions available for prediction when decoding each layer. Each time a prediction is created, the layered encoder must generate each decoder's imperfect prediction. By subtracting the decoded prediction from the original prediction, the exact amount of drift incurred is computed. Since drift tracking may be computationally and memory intensive for the layered encoder—especially for encodings with many layers—a simplification of this approach is to simply track the drift occurring at a single layer. For instance, decoders recovering only the base layer can be expected to incur the most drift, since the most symbols are dropped. Therefore, tracking the precise drift at the base layer gives an upper bound for the overall drift occurring by any decoder in the system.

Once a layered encoder is aware of the amount of drift, a mitigation strategy must be deployed to compensate for the drift or eliminate it once its amplitude exceeds a threshold. Drift compensation might work by having the layered encoder strategically recompute the residual for select pixel blocks in order to account for the accumulated drift experienced by lower-layer decoders at these locations. If the exact drift amount is known for decoders of each layer, then separate, unique drift compensation amounts could be computed for each decoder according to how much drift has occurred. This would have the effect of suddenly correcting the reconstruction of blocks for which drift has gotten too high.

If the exact amount of drift is not known, but instead the presence of drift is being tracked at each coding unit (i.e. the bit field strategy given above), then an IDR could be inserted dynamically by the layered encoder once a predefined percentage of the overall frame has been affected by drift. Inserting an IDR would require the layered encoder to know the full quality pixel-space reconstruction at that point, and would therefore require the presence of a full decoder which could be deployed to recover the pixel-space ground truth values when an IDR is determined to be necessary.

**Block-Wise IDRs**

Blocks of pixels represented in the encoded bit stream without using predictive coding are by definition immune to drift. This key insight motivates the final strategy proposed in this section for drift management. Instead of expending resources tracking and compensating for drift, the idea is to simply expect drift to occur and focus on limiting its effect by strategically coding blocks without prediction. A layered encoder could forcefully refresh selected blocks of pixels by encoding them in the base layer using non-predictive coding techniques. Once a block of pixels is selected to be refreshed, the layered encoder first queries an accompanying full decoder's complete pixel reconstruction to obtain the block's true pixel values. Next, the block of true pixel values is coded using lossless non-predictive coding instead of whatever coding mode was originally selected for the block. In H.265/HEVC, blocks of pixels can be signaled without prediction using `IPCM`

mode. Blocks signaled in this way will be received at full quality by all decoders, regardless of layer or existing drift in their DPBs. Thus, the proposed technique identifies certain blocks to be coded in the base layer in `IPCM` mode, thereby "stamping out" existing drift in the encompassed region of the frame for all decoders.

With this approach, layered encoders can control how much drift is allowed to propagate based on what percentage of blocks in each frame are switched to non-predictive `IPCM` mode. Presumably, blocks in each frame are selected either at random or according to a predetermined repeating pattern so as to eventually refresh all blocks after enough frames pass. Since `IPCM` mode generally costs more bits to encode the same pixel data when compared to predictive coding, there will be a bit rate overhead associated with stamping out drift. Thus, layered encoders must trade off between producing an adaptation with higher bit rate and higher drift, or an adaptation with lower bit rate and lower drift. This trade off should be optimized to maximize overall rate-distortion performance.

### 5.9.2 Novel Predictive Codec Design

Existing widespread video codecs are highly predictive and designed primarily to minimize compressed bit rate, particularly for large resolution content. As evidenced in this chapter, adapting highly predictive video to heterogeneous requests is challenging. In the long term, entropy-based content adaptation might work best with video data coded specifically with content adaptation in mind. Therefore, one important future work item identified by this dissertation is to design a new generation of video codecs that uses predictive coding to form a compact representation, but also provides a flexible representation to better support top-down adaptation. The sections below suggest a few speculative techniques which could help achieve this ambitious goal.

#### Limited Prediction Basis

An adaptation-friendly predictive codec must carefully manage the data dependencies caused by the recursive structure of predictive coding. Predictive coding requires that a set of known

previously signaled data is available for reference. To support predictive coding and adaptation at the same time, one strategy is to initially signal a small amount of data—perhaps a few blocks of pixels or a frame—to all receivers. This data is intended to be used heavily in future prediction. To avoid prediction drift, receivers are not allowed to adapt this content, and must receive it in full. Next, subsequent frames are represented in the compressed stream by referencing the small initial content basis. Receivers may adapt any aspect of these frames (including prediction syntax elements, as suggested below), since they are guaranteed to not be referenced in future prediction.

**Prediction Adaptation**

Another idea for novel codec design is to explicitly support prediction adaptation. In the ideal case, the prediction instructions for a given non-reference frame should be adaptable by smart receivers. For instance, the codec should support requests that specify a desired resolution for the encoded frame partitioning regime, or a desired precision for the encoded motion vectors. The encoded prediction syntax should be designed to be amenable to this type of adaptation. For natural video, prediction is generally an intrinsic trait of the scene. In other words, the motion vectors and frame partitioning information reflect underlying physical characteristics of the captured environment that are invariant to bit rate or other parameters of compression. Thus, each scene has an intrinsic ground truth prediction which should be adaptable on a per-receiver basis.

### 5.9.3 Convergence of Layered Probability Models

The layered extension of entropy-based content adaptation allows each layer to adapt its own probability model based on the symbol values successfully encoded by the layer itself and the symbols encoded in lower layers. Symbols encoded in higher layers cannot be used to inform a given layer's probability model, however, since those symbols are not necessarily available by a partial decoder. Assuming a statistically consistent symbol source, a context-adaptive probability model will eventually converge on the correct probability distribution, once enough symbols have been encoded. Since the highest layers are aware of a higher quantity of encoded symbols, they

are able to adapt their models more quickly to the content source. However, all layers' models will eventually converge, once enough symbols are encoded in the lower layers. This opens up interesting research questions about the nature of layer convergence and its effect on rate-distortion performance. This section describes some of the expected behavior while leaving a more complete analysis to future work.

**Effect on Compression Performance**

Entropy coding yields an optimal lossless compressed representation if an accurate probability model is used that correctly reflects the content source. For probability models that adapt to the source as symbols are encoded, this means early symbols are coded at lower efficiency, and later symbols are coded at near-optimal efficiency once the model has converged. For layered entropy coding, each layer's probability model converges to the optimal distribution at a different rate. In general, higher layers, which see a higher percentage of symbols, will converge more quickly. Lower layers see a lower percentage of symbols, and therefore take longer to converge. Eventually all layers will converge if the source statistics remain constant.

This suggests interesting research questions about how to optimize for compression performance given certain known source content traits. For example, suppose a particular symbol source is known to produce symbols according to a consistent probability distribution. One layering strategy for such a situation is to initially encode all symbols to the base layer. Base layer symbols will be received by every decoder, allowing all layers to adapt their probability models to the content. Once the rate controllers detect that an accurate probability distribution has been obtained, they can begin adapting the content by encoding symbols in higher layers. Assuming the source probabilities do not change, this will result in optimal coding for all symbols in all layers. The penalty of such an approach is that initially, symbols cannot be adapted and must be received in the base layer to provide the initial basis for probability model convergence.

Real-world video content is rarely statistically consistent for long. Scene transitions, new objects entering the field of view, camera panning, etc. all result in changes to the underlying source

distribution. As these changes happen, probability models need to constantly adapt to the new distribution. Therefore, one important area of research will be to explore strategies for layering symbols as source distributions change. For example, suppose a rate controller is able to detect content changes that manifest as changes to the underlying source probability distribution. Would it be preferable to immediately divert symbols to the base layer, in order to allow all layers to adapt their models to the new distribution? Or is it best to continue adapting symbols to different layers, with the expectation that eventually the coding efficiency of symbols encoded in lower layers will catch up? The answers to these questions depend on how stable the underlying source content is—that is, how likely it is to suddenly change its distribution—as well as how important adaptation is relative to compression performance.

**Analysis on Real Data**

Many of the anticipated behaviors described in the previous section could be demonstrated by generating a stochastic "fake" data source to produce symbols according to a hidden distribution. Next, layered entropy encoding could be used to encode the symbol stream with adaptive probability models in each layer. Finally, this experimental architecture could be used to empirically test sudden changes to the source distribution, different layering adaptation strategies, etc. However, ultimately these strategies should be tested on real-world data. M-JPEG data in particular is well-suited as a content domain for these experiments. For example, video content exhibiting a scene transition between two separate, relatively stable scenes could be used as the source. Next, the M-JPEG quantized coefficients that constitute the scene could be layered, allowing each layer's probability model to adapt separately to the content. Finally, strategies for scene transition detection and distribution adaptation can be assessed. The entropy-based layering encoding can be compared against a non-layered baseline, where a single probability model is able to adapt freely to every encoded symbol. We anticipate that clever model adaptation strategies can induce near-optimal compression performance, even when symbols are layered.

### 5.9.4 Application to Other Content Domains

Finally, this dissertation did not explore applications of entropy-based adaptation outside the domain of digital video. One of the most exciting traits of the proposed approach is that it is content-agnostic and can be used to adapt data from any content domain. While we leave to future work the task of applying entropy-based content adaptation to new domains, this section speculates about interesting possible applications that could benefit from such an approach.

#### One-Dimensional Digital Signals

One interesting application of entropy-based content adaptation is time domain, one dimensional, digitally sampled signals such as audio, temperature, voltage, or brightness data. This format is particularly amenable to symbol dropping because missing samples can be interpolated by the decoder and do not affect interpretation of future data. Suppose an arithmetic encoder is configured to directly encode samples to a compressed bit stream using a sophisticated context-adaptive probability model. Such a system could easily be modified to encode the samples into different layers according to indicator decisions made by a rate controller. Partial decoders which do not receive all layers can simply interpolate missing samples that were not coded in the layers they receive. Furthermore, more sophisticated rate controllers than the simple "bit budget" controller implemented in this chapter are possible. For instance, one idea is to implement a rate controller that keeps a memory of recent sample values, and elects to drop samples when the signal remains relatively constant and the missing values could be easily interpolated. Whether such a scheme could outperform a simple predictive or differential coding strategy is unknown, and would be an important research question.

#### "Frameless" Video

Typically, pixels in a video are spatially grouped into macroblocks and encoded as quantized frequency coefficients. However, each pixel can instead be interpreted as a one dimensional signal which is sampled over time. Thus, the ideas described in the prior section could be equally applied

to video data as well. The envisioned video codec works by entropy coding pixel intensities or prediction residual values directly from pixel space, without applying a frequency transform first. Thus, the rate controller is making symbol dropping decisions for individual pixel values instead of for quantized frequency coefficients. Each frame is therefore coded as a sparse matrix of pixel values, where some pixel values are present and others are dropped based on the rate controller's indicator decisions. In other words, pixel value updates appear to be "frameless" to the decoder, in the sense that they are not synchronized across every frame.

The proposed paradigm raises interesting questions about how a suitable rate controller could decide whether a pixel's value should be encoded or dropped in order to save bits at minimal quality loss. For instance, natural video often contains large regions of relatively unchanging background pixels. In these background regions, pixel intensity values may not need to be updated very often. Thus, a clever rate controller may be programmed to detect background regions and drop most pixel values in those regions. Dropped pixel values at the decoder might be simply extrapolated from prior pixel values in time, or interpolated between nearby pixel values in space. And if the codec supports prediction, the interpolation or extrapolation process for missing pixel values could be anticipated by an encoder in order to avoid drift. On the other hand, pixels in regions where motion is occurring will likely need to be updated in every frame. Motion can easily be detected by a rate controller if they have non-zero motion vectors or rapidly changing pixel intensities. For these regions, a smart rate controller could elect to encode all pixel values, ensuring that the content in these areas will be correctly recovered by the decoder.

# CHAPTER 6

## Drift-Controlled Residual Requantization

Chapter 5 presented a novel, fundamental approach where adaptation is cast as a symbol dropping problem during entropy coding. The primary strength of this strategy is its generalizability; it can be used to adapt any abstract symbol stream, regardless of content type. The key to obtaining a "good" adaptation for a given stream lies in designing an appropriate rate controller algorithm for the use case.

However, as evidenced by Chapter 5, applying layered entropy coding to the symbols produced by a standard hybrid video codec is challenging, due to friction between the hybrid and entropy prediction loops, as well as the tight inter-dependence between sequential symbols. Since the prediction loops are not unified, any symbol produced by H.265/HEVC's hybrid coder that is not included in the base layer contributes to drift in decoders that do not receive it. The drift produced by dropping symbols eventually dominates the reproduction, especially in low bit rate decoders, due to error propagation from temporal prediction. As evidenced, drift can be mitigated by incorporating more frequent IDR points or eliminated completely by switching to a non-predictive coding strategy. In the future, new predictive codecs may be designed that are better suited to context-adaptive entropy coding. But in the meantime, the end result is the same: much of the compression savings offered by predictive coding are lost by entropy-based adaptation.

This chapter takes a different, content-specific approach that integrates directly into the prediction loop of hybrid codecs. Instead of using a non-predictive codec or deploying drift mitigation strategies, as was investigated in Chapter 5, this chapter proposes a codec-specific method of tracking and compensating for drift by the encoder at request time. The proposed transcoding approach is similar to guided transcoding, but without the use of transcode-guiding

metadata. The key idea is to reuse the original prediction directly from the source encoding, and adapt the residual signal according to the client's needs. To prevent drift, the encoder implements a drift tracking mechanism which computes the drift incurred whenever the residual is adapted. The drift is recorded and compensated when predicting for future frames.

For encoded H.265/HEVC streams, a this strategy results in a much smoother rate-distortion curve and better overall performance than the content-adaptive entropy approach did. Although the transcoding operation still does not produce an optimal rate-distortion trade-off, transcoding complexity is reduced to the equivalent of a decoder, which greatly improves the system's ability to scale with the number of requests made.

## 6.1 Introduction

The system proposed in this chapter is a homogeneous H.265/HEVC fast transcoder. Instead of completely decoding the encoded video to pixel space and re-encoding it from scratch, as a cascaded transcoder would, fast transcoders attempt to re-use parts of the original encoding to speed up transcoding time. The proposed fast encoder directly reuses the complete, unaltered prediction instructions from the source encoding, and recomputes the residual to meet the requested target bit rate. Reusing the prediction is beneficial because it speeds up transcoding time by bypassing computationally expensive prediction searches. It also means that the original semantic model for the scene content is retained, even in lower-quality streams.

The transcoder begins with a full decoder to obtain the complete pixel-space reconstruction of the source video. This is important so that decoder drift can be tracked and compensated. The prediction syntax elements from the source encoding are passed through unaltered to the output bit stream, while the residual syntax elements are recomputed for the output bit stream to hit the target bit rate. However, every modification is tracked in pixel space so that the true reconstruction error is known by the transcoder. The transcoder therefore contains a second picture buffer to store the degraded reconstructed frames. When the prediction data for future frames is read from the source bitstream, the transcoder uses it to make two versions of the prediction: one using the true decoded

picture buffer, and one using the degraded decoded picture buffer. The former represents the source prediction, and the latter represents the transcoded prediction. Since both prediction signals are known, the transcoder is able to compensate for the prediction error at the decoder by recomputing the residual signal in the output bitstream.

Since the transcoder is already recomputing the residual signal to account for drift from prior frames, it can also requantize the adjusted residual to hit the lower target bit rate. In this way, by requantizing and recomputing the residual signal, the proposed fast transcoder is able to account for drift and change the output bit rate at the same time. The trade-off for these benefits is that the transcoder is required to contain a full decoder and an extra decoded picture buffer (DPB). This adds memory and complexity requirements that are absent in drift-prone solutions, but results in a much higher quality reconstruction.

A major guiding principle of this approach is to reuse the entire, unaltered prediction signal directly from the source instead of recomputing or altering it during transcoding. In contrast, full cascaded transcoding algorithms contain a complete encoder and therefore have to perform the computationally expensive task of searching for the optimal prediction signal. Additionally, remaining faithful to the original prediction signal ensures that any beneficial semantic understanding of the captured scene embedded in the prediction will be passed along to the transcoded version of the video. This is particularly important for codecs that have expressive, sophisticated prediction syntax, such as HEVC.

## 6.2 Hierarchy of Syntax Elements in H.265/HEVC

H.265/HEVC is a hybrid prediction-transform video codec. This means each frame is encoded in the bit stream in two parts: a set of prediction instructions and a set of prediction residual quantized coefficients. The prediction and residual data are represented by a series of *syntax elements* and encoded to the bit stream using entropy encoding techniques. To reconstruct the frame, the decoder decodes these syntax elements and uses them to create a prediction plane

and a prediction residual plane for the frame. The frame reconstruction is then produced by adding together the pixel values of the prediction and residual planes.

The technique proposed by this chapter involves separating the prediction and residual signals during transcoding, so that the residual data can be recomputed to achieve a lower target bit rate. Thus, the sections below describe the high-level H.265/HEVC syntax for coding partitioning, prediction and residual information.

### 6.2.1 Partitioning Syntax Elements

H.265/HEVC performs prediction and residual calculation on small blocks of pixels at a time. Thus, the first step in signaling a frame is to partition the frame into small coding units (CUs) that can be more easily predicted and transformed. First, the encoder and decoder split the frame into a grid of square coding tree units (CTUs). The encoded stream specifies whether the base CTU size should be 16x16, 32x32, or 64x64 pixels. Next, each CTU is split independently into smaller CUs using a quadtree structure. The CU quadtree structure for a given CTU is directed in the encoded stream by a series of dependent binary *split decisions*. Each split decision indicates whether a specific CU should be recursively split. A CU of size 8x8 is no longer allowed to be split, making 8x8 the minimum CU size. Encoders control how small the CUs are by choosing when to split them.

### 6.2.2 Prediction Syntax Elements

Once partitioning information has been signaled, the prediction for a frame is described on a per-CU basis. Prediction syntax elements are grouped by the following categories:

- **Prediction Mode** – Each CU is predicted using either inter-frame or intra-frame prediction. The prediction mode is signaled as a single binary decision for each CU. I-frame CUs do not require a prediction mode decision, since only intra-frame prediction is possible.
- **Prediction Unit Structure** – A CU block may be further split into smaller rectangular regions called *prediction units* (PUs) for prediction. PUs enable more precise prediction

along non-square boundaries. 8 different PU shapes are supported. The chosen shape for a particular CU is encoded as a sequence of binary decisions in the bit stream.

- **Advanced Motion Vector Prediction (AVMP)** – For each inter-coded PU, a motion vector is specified using a separate prediction loop, called *Advanced Motion Vector Prediction* (AMVP). The encoder and decoder deterministically formulate a list of *candidate motion vectors* to serve as the prediction for the block's eventual motion vector. The best candidate is specified in the bit stream as an integer index, and is used as the predicted motion vector. Finally, x- and y-direction offsets are coded in the bit stream for correcting the predicted motion vector coordinates.

- **Merge Mode** – As an alternate to signaling complete motion vector data with AMVP, inter-coded PUs may instead directly re-use motion prediction information from previously signaled temporally or spatially adjacent PUs. This saves bits in the encoded stream, particularly in regions of homogeneous motion where all motion prediction is roughly the same. The technique is called *merge mode*, and is indicated by a binary decision before the motion vector is specified. If merge mode is selected, an integer is also coded to indicate which nearby PU should supply the motion prediction information.

- **Reference Frame** – For each inter-coded PU, the correct reference frame to use for prediction must be indicated in the stream. This is coded as an integer, representing the index of the reference frame to use within the DPB.

- **Intra Prediction Mode** – For each intra-coded PU, an integer is encoded in the stream to indicate which intra prediction mode to use for predicting based on pixels from neighboring previously reconstructed blocks. H.265/HEVC has 35 intra prediction coding modes.

### 6.2.3 Residual Syntax Elements

After the prediction has been formed for the pixels in a given CU, the prediction residual for that CU is signaled in the encoded stream as a series of transformed, quantized frequency

coefficients. The residual information indicates the error incurred by the prediction, such that adding the prediction and the residual together reconstructs the pixel values. Residual syntax elements are grouped in the following categories:

- **Transform Unit Structure** – Just like with prediction, CUs may be partitioned into smaller blocks for frequency transform. The supported *transform unit* (TU) sizes are 32x32, 16x16, 8x8, and 4x4, and are specified using a quadtree structure with root at the CU level. Each *residual quadtree* (RQT) split decision is encoded as a binary decision in the bit stream. Note that all TUs must be square, due to the fact that the 2-dimensional DCT and DST must operate on square matrices. DST is only allowed for 4x4 TUs.

- **QP Value Offset** – After the frequency transform is applied to the prediction residual pixels, the resulting frequency coefficients are quantized. The amount of quantization is controlled by a *quantization parameter* (QP) value between 0 and 51, where higher numbers indicate higher quality. While a global QP value is specified in the encoded bit stream for each frame, each CU has the option of specifying a QP offset so that quality can be adjusted on a per-coding-unit basis.

- **Quantized Coefficient** – Finally, the quantized residual coefficients for the TU are encoded in zig-zag order, starting from the top-left-most coefficient and snaking around to the bottom-right-most coefficient. By placing the coefficients in zig-zag order, the earliest coefficients represent the lowest frequency components. The exact encoding of quantized coefficients is explained in greater detail in Chapter 5.

## 6.3 Overview of Approach

The proposed architecture is a homogeneous H.265/HEVC transcoder with decoder-like complexity. The transcoder adjusts the source stream to meet the target transcoded bit rate while accounting for decoder drift by recomputing and requantizing the residual data for each encoded frame. The following subsections outline the required components and operation procedures.

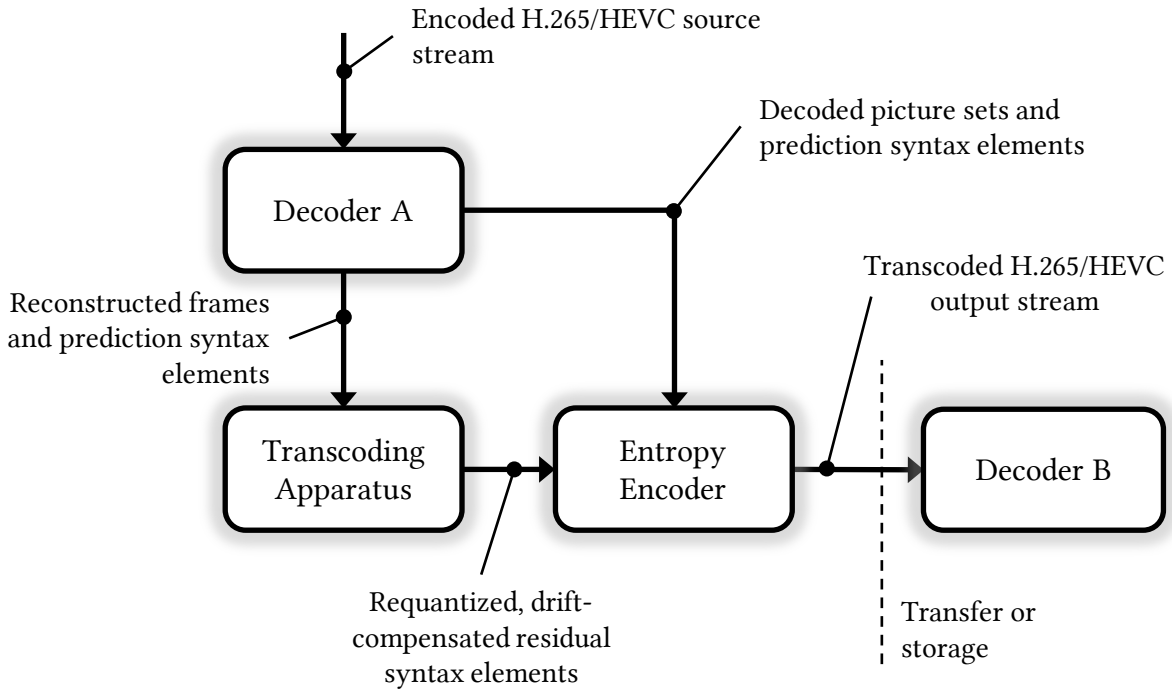



Figure 6.1: This block diagram depicts the components contained by the proposed bit rate fast transcoder. Decoder A represents a full H.265/HEVC decoder which decodes the source stream. The transcoding apparatus and entropy encoder blocks control transcoding and formulation of the output bit stream, respectively. Decoder B represents the endpoint client-side decoder and is a full H.265/HEVC decoder.

### 6.3.1 System Components

Figure 6.1 depicts the components envisioned for this system. Four connected components are included in the schematic: Decoder A, a transcoding apparatus, an entropy encoder, and client Decoder B. These components are briefly described below.

1. **Decoder A** – Decoder A is responsible for decoding the input source stream, producing the reconstructed pixel-space frames as well as the decoded syntax elements that were used to recover the frames. Since Decoder A must expose the entropy-decoded syntax elements to the rest of the system, most hardware-based decoder implementations are not suitable for this step.

2. **Transcoding Apparatus** – The transcoding apparatus modifies the syntax elements from the source stream by requantizing the residual to hit the desired bit rate target. To com-

pensate for drift, this component contains a second prediction-transform loop—including a separate decoded picture buffer—which is independent from the one encapsulated by Decoder A.

3. **Entropy Encoder** – Ultimately, the transcoded syntax elements are re-encoded using an entropy encoder such that another fully-compliant H.265/HEVC bit stream is produced as output. The entropy encoder component consists of a CABAC with independent context models and a static variable length coding (VLC) unit.

4. **Decoder B** – The transcoded stream produced by the entropy encoder is stored or transferred to the client, represented by Decoder B. Decoder B decodes the stream to recover a reproduction of the video at the requested lower bit rate. No drift occurs at Decoder B due to drift compensation performed by the transcoding apparatus.

### 6.3.2 Operation Procedure

The transcoding procedure contains two decoders, Decoder A and Decoder B, separated by a transcoding apparatus and an entropy encoder. The procedure is driven by Decoder A, which decodes the input source stream one syntax element at a time. Decoder A produces the complete, original, pixel-space reconstruction of each frame, and exposes the syntax elements it decodes to be used by the rest of the system. Since off-the-shelf hardware decoders do not typically expose the decoded syntax elements, Decoder A is likely to be software-based. Decoder A's operation is independent from and unaffected by later changes made to the stream by other components of the system.

Some of the syntax elements produced by Decoder A are forwarded directly to the output stream without alteration. This includes all syntax elements associated with frame prediction and parameter sets. But the residual signal is entirely recomputed by the transcoder apparatus in order to hit the lower target bit rate. As a result, Decoder A and Decoder B recover different pixel values for each frame. As explained in Chapter 5, any difference in the reconstructed frame between Decoder

A and Decoder B would normally introduce drift at Decoder B, due to the discrepancy between the prediction loops. The transcoder apparatus is therefore designed to track and compensate for these differences when computing each frame's residual. This process is described next.

For the $i$th encoded frame $F_i$, let $F_A$ be the pixel values recovered by Decoder A and let $F_B$ be the pixels recovered by Decoder B. $F_B$ is represented in the transcoded stream using the same prediction syntax elements that were originally used to code $F_A$. Let the prediction plane produced by Decoder A be $P_A$, and let $P_B$ be the prediction plane produced by Decoder B using the same prediction syntax elements. In the steady state, the reference pictures stored in the DPBs of Decoder A and Decoder B are different, due to the fact that Decoder B receives frames at a lower bit rate. Therefore, $P_A$ and $P_B$ will differ even though the prediction syntax elements used to form them are the same.

To prevent drift, the transcoding apparatus maintains a copy of Decoder B's DPB and uses it together with the prediction syntax elements to calculate $P_B$. The prediction error $E$ is the difference between Decoder B's prediction and the true reconstructed frame supplied by Decoder A.

$$E = F_A - P_B \tag{6.1}$$

Since Decoder B's DPB was used to formulate $P_B$, this means $E$ includes both the original residual data for frame $F_A$ plus any drift incurred due to imperfect reference frames at Decoder B. Therefore, introduce $D_B$ to represent the drift incurred by Decoder B due to imperfect prediction. The following equation describes the components of $E$:

$$E = R_A + D_B \tag{6.2}$$

Substituting Equation 6.2 into Equation 6.1 and solving for $D_B$ yields

$$D_B = F_A - R_A - P_B \tag{6.3}$$

But since $F_A - R_A = P_A$, this simplifies to

$$D_B = P_A - P_B \tag{6.4}$$

Thus, the transcoding apparatus can easily calculate the drift incurred by Decoder B due to imperfect prediction by computing the difference between Decoder A's prediction and Decoder B's.

Once the transcoding apparatus has computed Decoder B's prediction error $E$ via Equation 6.1, it transforms and quantizes $E$ using the appropriate QP value required to hit the target lower bit rate. This is termed *requantization*, since the component of the signal represented by $R_A$ in Equation 6.2 was previously quantized in the initial encoding, and now is quantized again after factoring in the drift component $D_B$. It is assumed that the new QP value is selected by an external rate control algorithm to achieve the requested bit rate. To reduce transcoding time, the same residual quadtree and other structural information from the original encoding are simply re-used during requantization of $E$. Requantization is implemented by applying the standard transform and quantization routines from an H.265/HEVC encoder to the computed $E$ plane.

Additionally, the transcoder must also recover the complete reconstructed frame $F_B$ as produced by Decoder B so that $F_B$ can be added to the transcoder's DPB for reference in future prediction. Thus, the transformed and quantized coefficients are de-quantized and inverse-transformed to recover Decoder B's recovered version of the prediction residual, $R_B$. Finally, the transcoder computes $F_B = P_B + R_B$ and incorporates $F_B$ in its DPB.

In summary, the transcoder apparatus calculates the new residual in pixel space, producing a set of transformed, quantized coefficients which are sent to the entropy encoder to be multiplexed into the output bit stream. The final transcoding unit is an entropy encoder, which consists of a CABAC encoder with associated contexts as well as a variable-length coding (VLC) unit. As Decoder A process syntax elements, those associated with the video, sequence, and picture parameter sets (VPS, SPS, and PPS, respectively); prediction formation; and stream structure (i.e. NAL unit headers) are forwarded directly to the entropy coding unit. When it is time to encode residual

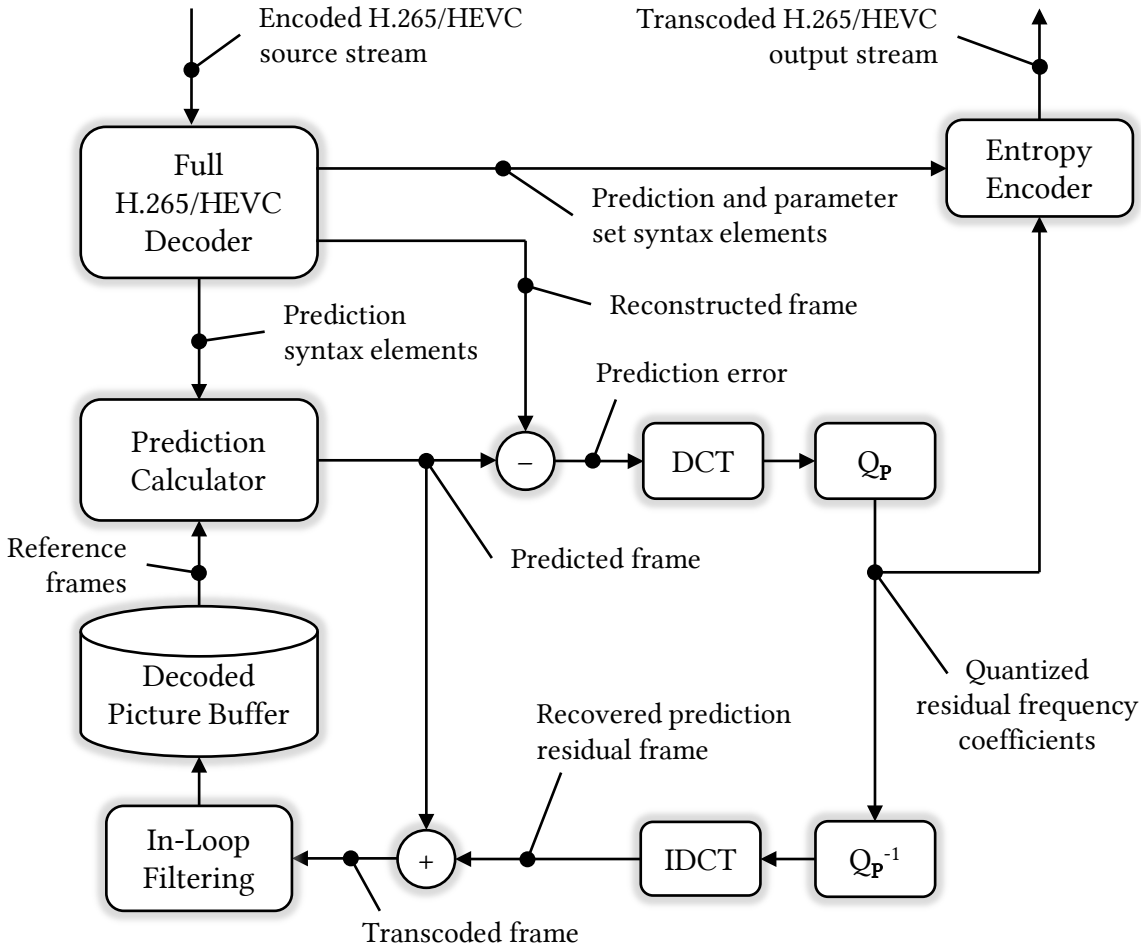



Figure 6.2: This block diagram depicts the complete proposed fast bit rate transcoder, which reuses the prediction directly from the source bitstream and therefore avoids performing the computationally expensive prediction search algorithm.

data, the entropy encoder switches its source to use the coefficients produced by the transcoding apparatus. The bit stream produced by the entropy coding unit is fully standard-compliant and can be decoded by any non-scalable H.265/HEVC decoder like the one represented by Decoder B.

### 6.3.3 Block Diagram

A complete block diagram depicting the proposed hybrid transcoder is shown in Figure 6.2.

## 6.4 Benefits of Proposed Approach

This section describes the high-level objectives achieved by the proposed approach, and compares them to alternate approaches.

### 6.4.1 Drift Compensation

Chapter 5 described the tension between layered coding and predictive coding. When the bit rate of a predictive-coded video is scaled, the encoder must either compensate for the missing information at the decoder during prediction, or allow decoder drift to be introduced in the system. By performing per-request drift compensation, the proposed approach is able to maintain the rate-distortion performance benefits afforded by predictive coding, while simultaneously ensuring that drift does not occur—all while allowing the decoder to dynamically scale the encoded bit rate. Thus, the positive traits of scalable coding and predictive coding are simultaneously achieved.

### 6.4.2 Decoder and Client Complexity

Another major benefit of the proposed approach is that the output bit stream is fully standard-compliant, and can be decoded by a hardware-based, non-scalable H.265/HEVC decoder. Other scalable approaches, including the layered approach taken by SHVC, require explicit decoder support and increased decoding complexity to receive and incorporate enhancement layers. This limitation is simply not tenable in an environment where high-quality video is dominant and should be prioritized for maximum compatibility.

Furthermore, by using the proposed approach, a receiver is able to tailor the transmitted bit stream exactly to its unique bit rate needs. Simulcast and layered approaches for generating a scalable encoding rely on pre-selecting a few discrete bit rates which are explicitly support. Decoders are allowed to select from the available rate options, but cannot ask for an encoding at a rate in between two neighboring options. On the other hand, the proposed approach allows receivers to request an encoding at any bit rate, and a suitable stream will be produced on-the-fly.

Furthermore, smart receivers can even change the requested bit rate dynamically during transfer. This could be used, for instance, in a situation where the receiver's network bandwidth noticeably fluctuates.

### 6.4.3 Encoder and Server Complexity

The benefits described above are made possible because the source bit stream is transcoded on-the-fly in response to a client's request. This obviously introduces a server load that scales linearly with the number of requests. The final section of this chapter discusses possible mitigation strategies for reducing the server load, including batching similar simultaneous requests and lazily caching the transcoded results for common requests. However, there will ultimately always be a server complexity overhead with all scalable approaches that use just-in-time transcoding.

Nonetheless, certain design decisions in the proposed approach result in reduced per-request server load when compared to a full cascaded transcoder. Most notably, the decision to reuse the same prediction syntax elements as the source eliminates the need to perform a prediction search. Overall, the per-request transcoding operation requires a full H.265/HEVC decode with the addition of an extra DPB, a residual requantizer, and an entropy encoder.

## 6.5 Justification for Prediction Reuse

With the proposed system architecture, all decoders receive identical prediction syntax elements, regardless of the bit rate that they requested. Among others, the following prediction attributes are universally sent to all decoders invariant of requested bit rate (see Section 6.2.2 for full details): frame partitioning structure, inter/intra prediction coding decisions, motion vectors, and intra coding modes. Since it is used in all output streams, the prediction structure has the potential to greatly affect coding efficiency across the entire bit rate spectrum. It is therefore important to use a "good" prediction. Otherwise, if an inaccurate or suboptimal prediction is used, coding efficiency would suffer across all requested bit rates.

But what constitutes a "good" prediction? Classically, the best prediction is the one that allows the encoder to maximize quality while hitting the target bit rate. This problem statement is quantifiable and compression-oriented, but it reflects the historical tendency to over-focus on rate-distortion performance. In the face of diversifying, heterogeneous clients, no longer is rate-distortion performance the only trade-off worth optimizing. This dissertation claims that client flexibility is of increasing importance in a diverse world of heterogeneous receivers. Coded prediction data conveys semantic information about the scene such as structural and motion information. In the future, modern smart clients may be interested in receiving this data at high quality, for sophisticated application-level goals. Therefore, this data should be included in response to all requests even if it affects rate-distortion performance.

The system architecture described in this chapter simply reuses the prediction syntax elements as-is from the source encoding of the input video. This is justified by the assumption that the original encoded prediction was captured by the source device, and therefore represents the ground truth of the scene. However, if for some reason the source prediction structure is unreliable, one could easily imagine a simple modification to the proposed system where a prediction search is performed as a one-time operation before any transcoding is performed. Although this prediction parameter search would be computationally expensive, it would be a one-time operation for each uploaded video and would not be performed per-request. The rest of the proposed system could then use the new prediction structure instead of the original one for transcoding.

### 6.5.1 Stream Composition

The proposed technique hits the requested target bit rate by requantization—that is, by reducing the precision of the quantized, residual frequency coefficients in order to save bits until the desired rate is reached. This enforces a lower bound on the achievable transcoded bit rate, since no prediction or parameter set information can be modified. The lowest possible bit rate achievable by the proposed technique would be to completely remove all residual information from the transcoded stream, and retain only the prediction and parameter set information (i.e. the encoding would have

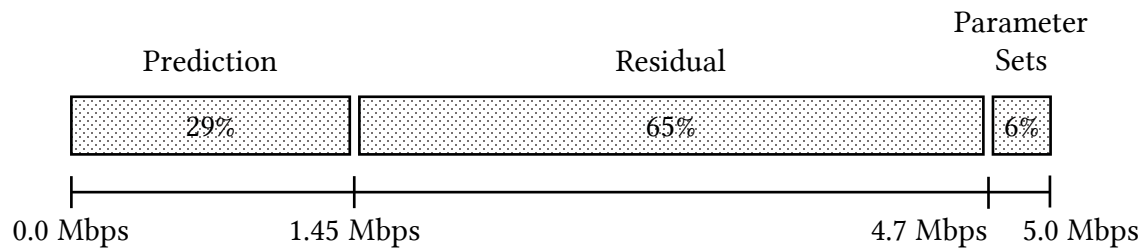



Figure 6.3: This figure depicts the average proportions of a 10s, 1080p H.265/HEVC stream encoded with x265 on profile "veryslow" at $5$ Mbps. Each bit in the encoded stream is labeled either prediction, residual, or parameter set based on its purpose. Thus, the values presented here represent the average across the entire video segment. Five different sample videos were encoded and their results averaged to produce this figure.

an *empty residual*). Note that this would result in a reconstruction of very low quality, consisting entirely of structural prediction data. It could be used, however, by clients who are requesting the stream primarily for the embedded motion or structural data.

Figure 6.3 depicts the composition of an average H.265/HEVC stream produced by the x265 encoder at $5$ Mbps. The encoded bits associated with prediction signaling account for 29% of the overall stream, those associated with residual signaling account for 65%, and those associated with parameter sets account for 6%. Requantizing such a stream can therefore achieve a minimum bit rate of $5$ Mbps $\times\ 35\% = 1.75$ Mbps and a maximum bit rate of $5$ Mbps.

However, most non-scalable H.265/HEVC encoders—including x265 and HM—determine stream composition based on the target bit rate. In other words, not all streams produced by these encoders are composed of 29% prediction bits, 65% residual bits, and 6% parameter set bits. In fact, when encoding at lower bit rates, x265 tends to use a higher percentage of the stream to signal prediction, and a comparably lower percentage to signal residual. As the target bit rate increases, x265 starts to allocate a higher proportion of the stream to signal the residual. This is illustrated by Figure 6.4, which depicts the bit stream proportions when sample video "Jellyfish" is encoded multiple times using x265 on "veryslow" profile at various bit rates.

Ultimately, the proportion of the bit stream that gets allocated to signaling prediction versus residual data is an encoding decision, and may vary greatly from stream to stream. Encoders that are configured to perform exhaustive prediction search will be time consuming, but will produce an

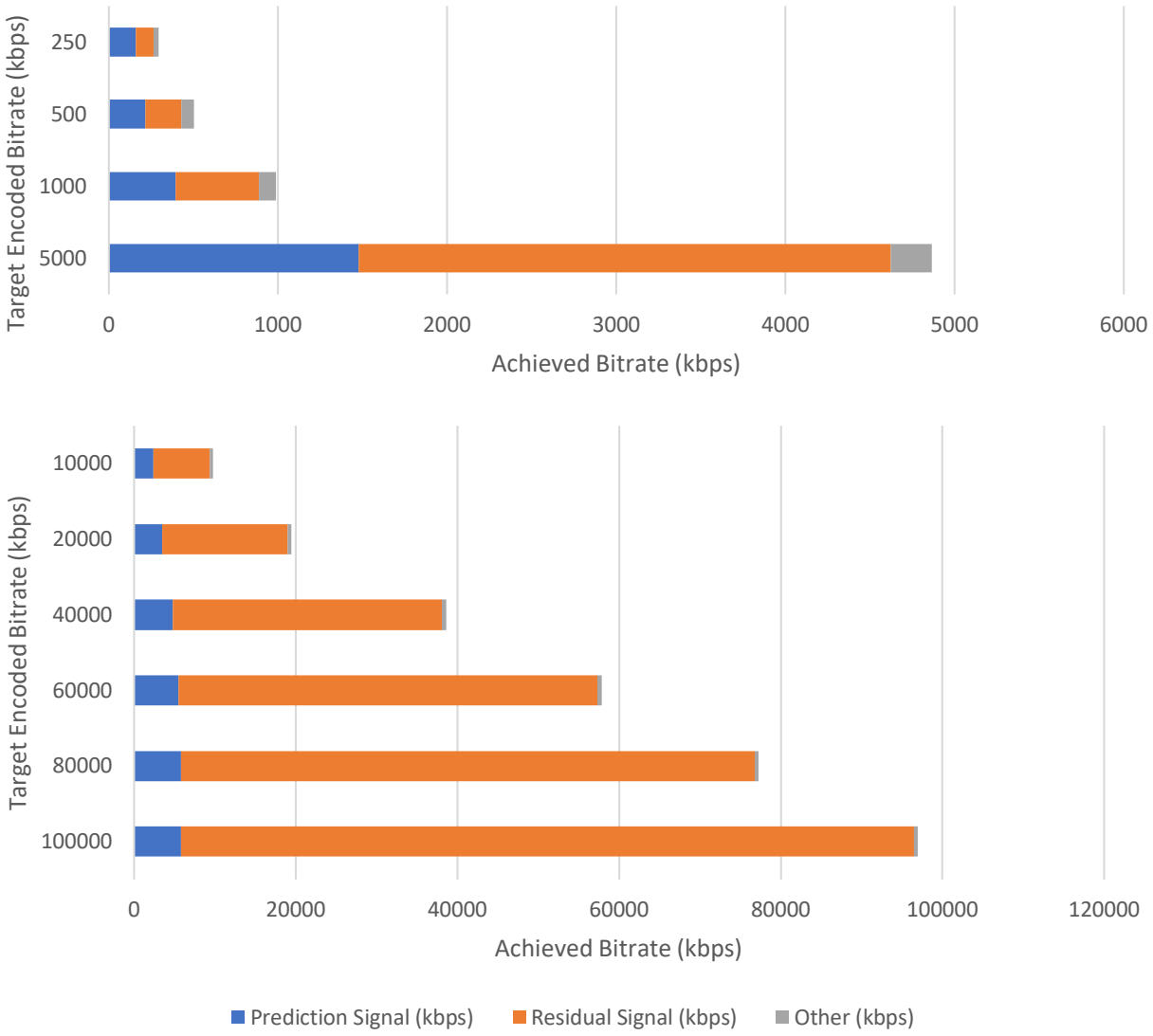



Figure 6.4: Proportion of bitstream allotted to prediction and residual signal when “Jellyfish” sample video is encoded using x265 H.265/HEVC encoder on “veryslow” profile at various bit rates.

output stream with high-quality prediction and structural information. On the other hand, encoders configured to take shortcuts by making quick decisions about prediction modes may not produce optimal prediction. These decisions ultimately affect the stream composition and reconstruction quality. In summary, prediction quality is a function of encoder complexity.

### 6.5.2 Source Encoding as Ground Truth

The motion vectors, spatial texture patterns, and frame partitioning information that constitute the prediction for an encoded H.265/HEVC stream represents a rich semantic understanding of the dynamics of the content. Someday, we envision that smart clients could be tuned to make high-level inferences about what is happening in the scene using prediction data alone.

Modern consumer-grade digital capture devices such as smartphones and digital video cameras record video using on-chip hardware encoders. These CCD or CMOS sensor chips produce pre-encoded H.264/AVC or H.265/HEVC content as the initial representation of the video data. In the future, hardware encoding will continue to become more common, meaning consumer-captured digital video data will likely spend its entire life cycle in compressed form. On the opposite side, decoding also increasingly takes place in hardware, as physically close to the display apparatus as possible. This suggests that the encoded representation of a consumer-recorded video is, in fact, the primary representation. Under this interpretation, the decompressed, pixel-space representation is merely an alternate rendering of the true, compressed-domain content. Furthermore, the initial encoding captured and produced by the source represents the ground truth for the video content, and all transcode operations for the rest of the video's life cycle represent degradations of that original encoding. Therefore, it is imperative when performing homogeneous transcoding to preserve the quality of the prediction as much as possible. The original prediction represents important semantic information captured by the source about the scene.

Since the prediction information embedded in a compressed video reflects scene semantics, it should be relatively invariant to changes in encoded bit rate. In other words, the motion, pattern, and structural information within a scene are fixed regardless of an encoder's compression rate. This is further justification for reusing the original prediction captured in a scene whenever possible. However, when encoding content to very low bit rates, encoders in practice tend to use simpler predictive structures in order to spend less bits overall on prediction. Since predictive structure is a function of the scene, not the bit rate, this suggests that the ideal lower quality prediction for a low

target bit rate can be deduced based on the higher quality prediction. The exploration of this idea is left to future work.

Finally, reusing the initial, highest-quality prediction for lower bit rate requests yields another benefit: it naturally results in a top-down scalable rate-distortion curve. Recall that a top-down scalable encoding is one that optimizes requests at high bit rates and allows rate-distortion performance to drop at lower bit rates. A top-down solution is increasingly desirable as the proportion of high bit rate requests continues to increase globally. By using the prediction originally associated with a high bit rate encoding, requests at high bit rates will result in near-optimal rate-distortion performance. Requests for lower bit rates will result in sub-optimal performance, since the prediction used will be overly precise for the requested bit rate.

## 6.6 Application to H.265/HEVC

Residual requantization is a content-specific approach designed to induce top-down, receiver-driven video adaptation for video content that was initially encoded using a tightly inter-dependent, predictive codec. To evaluate residual requantization, this section presents the results of an experiment where the technique has been applied to data initially encoded as a H.265/HEVC stream. Thus, the algorithm, results, and conclusions presented here are specific to H.265/HEVC coded content.

### 6.6.1 Residual Requantization of H.265/HEVC Data Streams

A residual requantization apparatus was designed as a modification of the HM reference implementation of H.265/HEVC. The module is essentially a homogeneous transcoder, such that pre-encoded H.265/HEVC data is provided as input, and a modified (but still standard-compliant!) H.265/HEVC stream is generated as output. The idea is to enable receivers to control the residual requantization process in order to specify the bit rate that they need on a per-frame basis (i.e. adaptation is receiver-driven). Furthermore, receivers with enough bandwidth to receive the entire

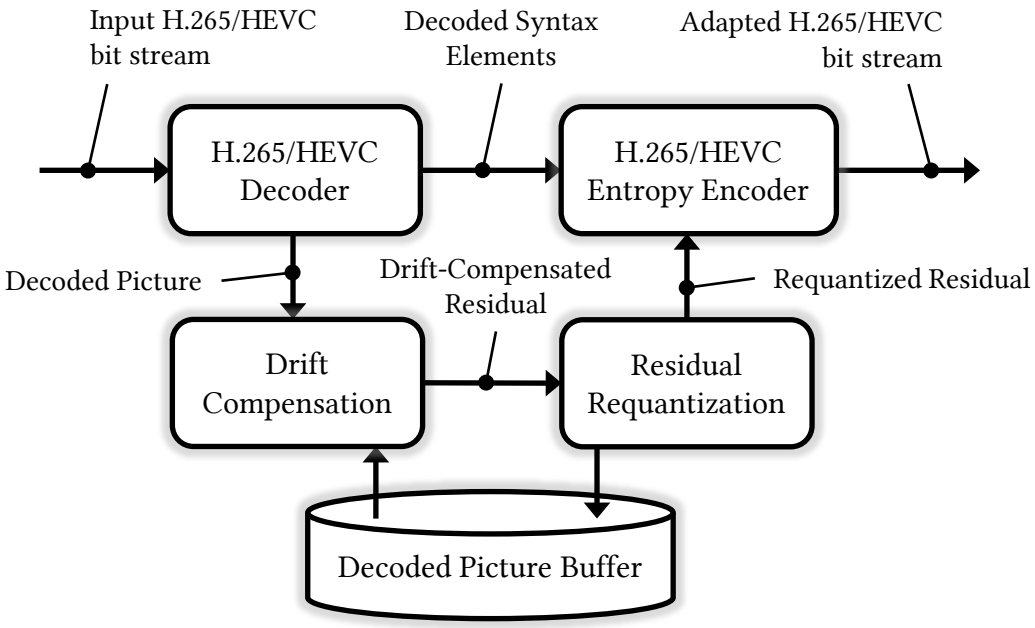



Figure 6.5: This block diagram depicts the high-level components of the proposed residual requantization transcoder.

stream should receiver the initial encoding (i.e. adaptation is top-down). Both of these ideals are realized by the implementation.

Figure 6.5 illustrates a system block diagram for the implemented transcoder. The initial H.265/HEVC bit stream is immediately decoded by a full H.265/HEVC decoder. During the decoding process, the decoder exposes the syntax elements it produces for each frame for the rest of the transcoder to use. Next, the decoded syntax elements associated with the residual signal are recomputed to hit a target bit rate supplied by the receiver for the given frame and to compensate for any pre-existing decoder drift that had been tracked by the drift estimator. The recomputed residual is then multiplexed back into the initial stream, entropy-encoded, and sent to the decoder. Finally, the drift estimator incorporates any drift produced by the requantization process back into its estimation for prediction of future frames.

The resulting system can be expected to exhibit the following behavior:

- **Drift-Compensation** – The drift compensation mechanism ensures that decoder drift does not affect prediction of future frames. Thus, reconstruction quality should not decrease over time.

- **Receiver-Driven Bit Rate Adaptation** – Receivers are able to request and receive the exact bit rate that matches their application needs and available network bandwidth. This can be specified on a per-frame basis.

- **Top-Down Adaptation** – Since the original prediction and structural information is directly re-used by the transcoder, adapted streams provide as much original information as possible. Receivers with high available bandwidth can receive the full stream as originally encoded.

### 6.6.2 Data Set

The results presented in this section were generated using the free, public-domain *Sunflower* video content. This content consists of 500 frames captured at 30 frames per second, with a resolution of $1920 \times 1080$ and 24 bits per color pixel. The initial H.265/HEVC "ground truth" encoding used for all experiments in this section was generated using ffmpeg's version of libx265 (Tomar, 2006; MulticoreWare, 2019) on the *veryslow* profile. A sample frame from the source *Sunflower* video is reproduced in Figure 6.6.

### 6.6.3 Results and Discussion

To evaluate residual requantization as an adaptive approach, it was applied to an input H.265/HEVC encoding of the *Sunflower* content described above. Receiver-driven adaptation was simulated by requantizing the content multiple times at various different bit rates. Next, the results were compared to a baseline adaptive approach where the source content is simply transcoded to lower bit rates using an off-the-shelf cascaded transcoder. In particular, ffmpeg was used for

Figure 6.6: This figure depicts a sample full-quality frame from the video content used for the H.265/HEVC residual requantization experiment.

the baseline approach with its implementation of the libx265 library (MulticoreWare, 2019). The default encoding profile was used for all baseline adaptive cascaded transcoding operations.

Comparing content adapted with ffmpeg's cascaded transcoder against content adapted with residual quantization provides a valuable benchmark for the rate-distortion performance of the proposed approach. However, note that there are a few benefits unique to residual requantization. First, it is a computationally faster approach than cascaded transcoding, because it does not recompute the prediction used to encode the content. Second, it offers top-down adaptation since receivers with high available bandwidth will ultimately receive an encoded stream that is quite similar to the initial encoding. Conversely, the baseline cascaded approach always throws away the prediction information from the original encoding, meaning no clients will receive this valuable representation of the captured content.

At the same time, one advantage of the baseline approach is that since it recomputes the prediction from scratch every time, it is able to achieve better rate-distortion performance at the low end of the bit rate spectrum. This is because the original ground-truth prediction is overly specific for low bit rates, and takes up too many bits in the encoded stream.

Figure 6.7 depicts sample high quality and low quality frames from both adaptation approaches. Clearly, both approaches are able to adapt content down to lower bit rates where quality loss becomes visually perceptible.

Figure 6.8 depicts per-frame bit rates for three different adaptations of the source content: one at 9.76 Mbps, one at 4.71 Mbps, and one at 3.57 Mbps. All of these adaptations were created by residual requantization. This figure indicates that the drift compensation mechanism integrated in residual requantization works correctly, and that drift is not a significant issue in the results. In all three adaptations, the quality is initially higher, and dampens slightly over time. This might be due to the fact that initially, the encoder and decoder picture buffers are exactly synchronized (i.e. both are empty). As frames are transmitted, the encoder and decoder DPBs begin to drift. Although the effect of drift is compensated, its presence may cause this slight quality performance dip.

Finally, Figure 6.9 compares the rate-distortion performance of residual requantization against that of the baseline cascaded transcoding approach. This graph illustrates a few interesting points. First, it demonstrates how residual requantization outperforms ffmpeg at high bit rates. This is by design, and provides strong evidence that top-down adaptation does indeed prioritize requests at the high end of the bit rate spectrum. Second, note that the default encoding profile of ffmpeg struggles to reach the highest range of video qualities. This is because it throws away the prediction from the initial encoding, which was quite precise, and instead tries to compute a new prediction on demand. Had the ffmpeg profile been set to compute a slow, multi-pass prediction, it might have been able to reach these higher bit rates. But that would be a silly approach: to throw away a good initial prediction and then spend extra time recomputing it.

Third, ffmpeg clearly outperforms residual requantization at the lowest band of bit rates. In fact, residual requantization is unable to reach the lowest bit rates at all, due to the high portion of its bit stream expended to signal the initial prediction information. This is known and expected behavior. However, signaling the original prediction could have secondary advantages over recomputing the prediction from scratch. For instance, providing the initial prediction gives receivers semantic

information about the captured scene that could be useful, such as enabling cheap optical flow inference from the embedded motion vectors.

(a) A sample high bit rate frame produced by libx265 through cascaded transcoding

(b) A sample high bit rate frame produced by drift-controlled residual requantization

(c) A sample low bit rate frame produced by libx265 through cascaded transcoding

(d) A sample low bit rate frame produced by drift-controlled residual requantization

Figure 6.7: This figure provides a visual comparison of sample frames produced by cascaded transcoding as well as the proposed residual requantization approach. Figures (a) and (b) depict high bit rate frames produced by the two techniques. Figures (c) and (d) depict low bit rate frames by both approaches.

## 6.7 Future Work

The experimental results presented in this chapter evidence that residual requantization is a viable domain-specific method for achieving top-down receiver-driven content adaptation. Nonetheless, this section identifies two avenues of inquiry for future work. First, residual requantization by itself is only able to maintain high rate-distortion performance at the high end of the bit rate spectrum. We therefore propose that by introducing prediction adaptation techniques, residual

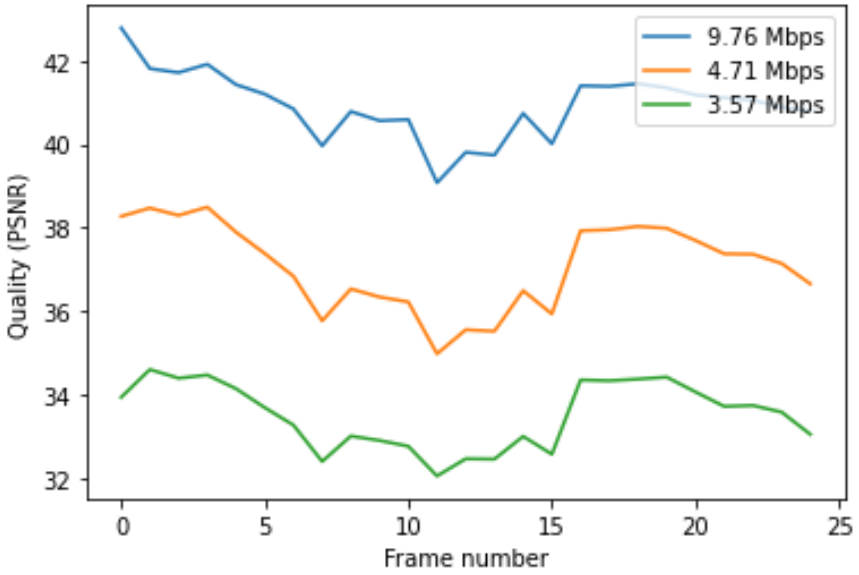



Figure 6.8: This figure illustrates the quality achieved for each frame of video adapted with residual requantization.

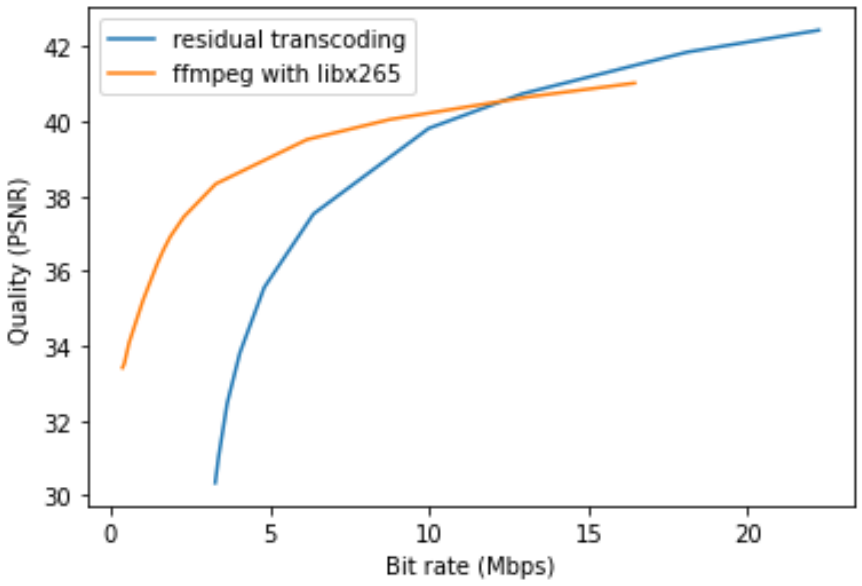



Figure 6.9: This figure depicts the rate-distortion performance achieved by residual requantization in comparison to cascaded transcoding with ffmpeg.

requantization can more effectively adapt to lower bit rates. And second, an untapped potential application for residual requantization is multiple description coding. We therefore propose an adaptation strategy where multiple independent low quality versions of the same video are created so that they can be combined together to produce a higher quality version of the content.

### 6.7.1 Prediction Adaptation

Since residual requantization does not modify the original prediction, it cannot adapt the compressed bit rate to be lower than the proportion of the original bit stream associated with the residual signal. Depending on how the source content was encoded, the prediction signal sometimes may account for a larger proportion of the overall bit stream than the residual. Thus, there may be situations in practice where a receiver wishes to adapt the source encoding below the minimum amount afforded by the prediction/residual composition of the original representation. To fulfill these requests, the prediction must be altered to accommodate the lower bit rate. For example, the frame partitioning structure might be made coarser or the motion vectors might be modified to use merge mode instead of AMVP. As presented, the basic residual requantization algorithm explored in this chapter does not adapt the prediction at all, and is therefore unable to accommodate these requests at lower bandwidth. This limitation is illustrated in Figure 6.9, where residual transcoding is unable to reach the lowest bit rates. As future work, we propose an extension to residual requantization where the prediction from the original encoding is adapted to meet lower bit rate targets, but remains true to the original captured content as much as possible. We term this proposed technique *prediction degradation*.

The idea behind prediction degradation is to adapt the originally encoded prediction instructions so that a receiver-driven target bit rate can be achieved. If the target bit rate is low, the prediction instructions should be made coarse enough to meet it. As much as possible, the adapted prediction should retain the structural details and semantic understanding of the original scene (i.e. it should remain "top-down"). A simple approach could be to identify certain motion vectors to be switched from AMVP mode to the less costly merge or skip mode. Good candidates for merge

mode could be identified based on detection of homogeneous motion across a region of the frame. Once a motion vector is switched to skip mode, the residual signal is recomputed and requantized like normal. Another idea is to adapt the partitioning structure used to code a particular CTU. Low bandwidth clients may not be able to afford the full detail provided by the quadtree of the original encoding. Instead, the quadtree could be pruned to save space in the adapted stream.

### Machine Learning Approach

The easy availability of ground-truth solution data for this problem coupled with the hypothesized coherence between the original prediction signal and its lower-bit-rate counterpart make it an excellent candidate for a machine learning approach. To this end, a series of prediction adaptation models would be trained to estimate the ideal adapted prediction coding parameters for a given block of source pixels. Each model would be designed to generate an adapted prediction for a specific lower bit rate target. For a given block and a given target bit rate, the model would predict the ideal adapted prediction mode (i.e. inter vs intra), prediction block sizes, and intra mode (i.e. angle) or motion vector (i.e. dx, dy). As input, the model would be given a patch of the reference pixels surrounding the block, the desired reconstruction for that block, and the prediction coding parameters originally used for the block in the source encoding. As output, it would produce an estimate for the correct prediction coding parameters for the block.

A large dataset for the proposed model could be generated by independently transcoding source video content to lower bit rates using cascaded transcoding. The prediction instructions embedded in the transcoded video would then be extracted and considered as the ground truth for the target bit rate. This process would be repeated on a variety of content representing different scene dynamics, frame rates, source bit rates, target bit rates, etc. Ultimately, a large corpus of video data would be produced. One suggestion is to organize the dataset as a collection of block-wise samples, each representing the adaptation decisions made by a cascaded transcoder for a particular block at a particular target bit rate. Each sample has the form $(r_0, r_1, R_0, B_0, P_0, P_1)$, where $r_0$ is the bit rate of the original encoding, $r_1$ is the target lower bit rate used by the cascaded transcoder, $R_0$

contains the reference pixel values originally used to predict the given block, $B_0$ contains the true reconstructed pixel values for the block, $P_0$ represents the prediction instructions used to code the block in the original encoding, and $P_1$ represents the adapted prediction instructions produced by the cascaded transcoder to code the block at target bit rate $r_1$. For samples where $r_0 \approx r_1$, one could expect that $P_0 \approx P_1$. However, if $r_0$ and $r_1$ are further apart, $P_1$ should begin to change, indicating that the prediction was adapted by the cascaded transcoder. Given enough samples, a sufficiently sophisticated model might be able to accurately predict $P_1$ given input values of $(r_0, r_1, R_0, B_0, P_0)$ for a specific block. The trained model could then be used to accurately and quickly adapt a video's prediction according to the receiver-driven target bit rate.

### 6.7.2 Multiple Description Coding

The intended use case for residual requantization is to better support receiver-driven, top-down adaptation. However, video adaptation is not the only application that could benefit from such an approach. One exciting example is multiple description coding. Multiple description coding is a well-known existing technique whereby video content is coded into multiple independent representations called *descriptions*. A decoder can use any individual description to approximate the original content. However, decoders that receive more than one description can combine them to reproduce a better approximation of the content. In general, the reproduction quality increases with the number of descriptions that a decoder receives and incorporates. Multiple description coding is very similar to layered coding (i.e. descriptions correspond to layers), with the caveat that every description must be independently decodable to produce an approximation of the content. Layered coding, on the other hand, maintains a dependency hierarchy between layers such that enhancement layers are not useful by themselves.

Residual requantization works by recomputing the residual signal to hit a receiver-specified lower bit rate target, while accounting for decoder drift. The implementation presented in this chapter lowers the compressed bit rate by increasing the quality parameter (QP) value evenly for all blocks in a frame. This results in a roughly even decrease in perceived quality across the entire

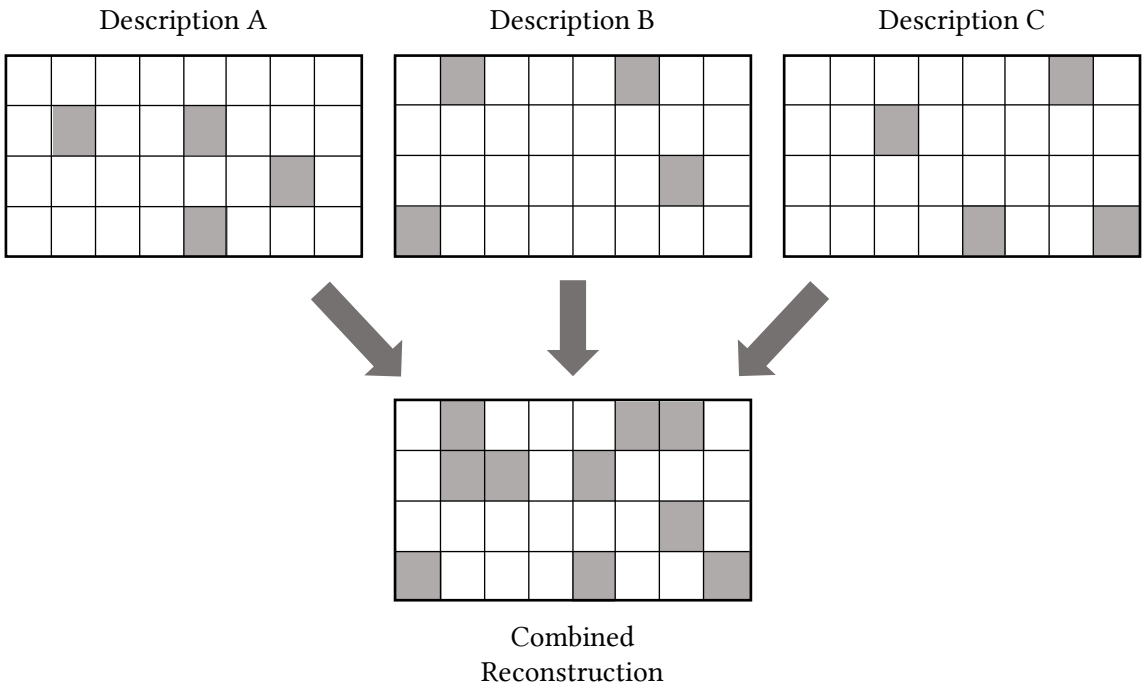



Figure 6.10: This figure depicts a possible application of residual requantization to achieve multiple description coding. The same video frame is depicted from three independent descriptions (descriptions $A$, $B$, and $C$). Square coding blocks are superimposed over each frame. White coding blocks were requantized at a higher QP value, and gray coding blocks were requantized at the same QP value as the original encoding. A decoder receiving all three descriptions can combine the gray coding blocks to produce a higher quality reconstruction.

frame, once the computed residual has been requantized with the higher QP value. However, residual requantization also is able to target loss to certain spatial regions, color channels, or frequencies of the residual signal. For example, instead of applying loss evenly, suppose loss is selectively applied to a small subset of the pixel blocks. For each encoded block of pixels, the transcoder determines whether loss should be applied according to a deterministic pattern. For blocks that incur loss, the QP value is increased before requantization is performed. For blocks that do not incur loss, no change is made to the QP value. The compressed bit rate can therefore be controlled based on how many blocks are requantized to a higher QP value.

This strategy can be used to generate multiple descriptions for the source content. The key is for the transcoder to vary the deterministic pattern used to identify which pixel blocks should incur loss when creating each description. Thus, each description is assigned a different pattern of

high quality and low quality blocks. If a decoder is aware of the deterministic algorithm used to select high quality and low quality blocks, then it knows which blocks were coded at the original quality in each description. Thus, it could combine multiple descriptions by unifying the high quality blocks from each description together in a single reconstruction. The unified reconstruction would have a higher overall quality than any of the individual descriptions. An example decoder recombination procedure is depicted in Figure 6.10.

Instead of targeting loss to certain blocks, an alternate strategy is to target loss to certain spatial frequency bands. This could be implemented by assigning a different custom quantization matrix to each description before applying residual requantization. Each description's custom quantization matrix would prioritize a different frequency coefficient or frequency band. Thus, a decoder that could identify the high-quality frequency bands from each description could extract them to produce a more accurate combined reconstruction of the overall signal.

Finally, note that in each of the proposed implementations of multiple description coding, the encoded descriptions represent fully standard compliant H.265/HEVC bit streams. Thus, the system is fully backwards-compatible in support of decoders that are not equipped to combine multiple descriptions together.

# CHAPTER 7

## Future Work: Syntax Element API

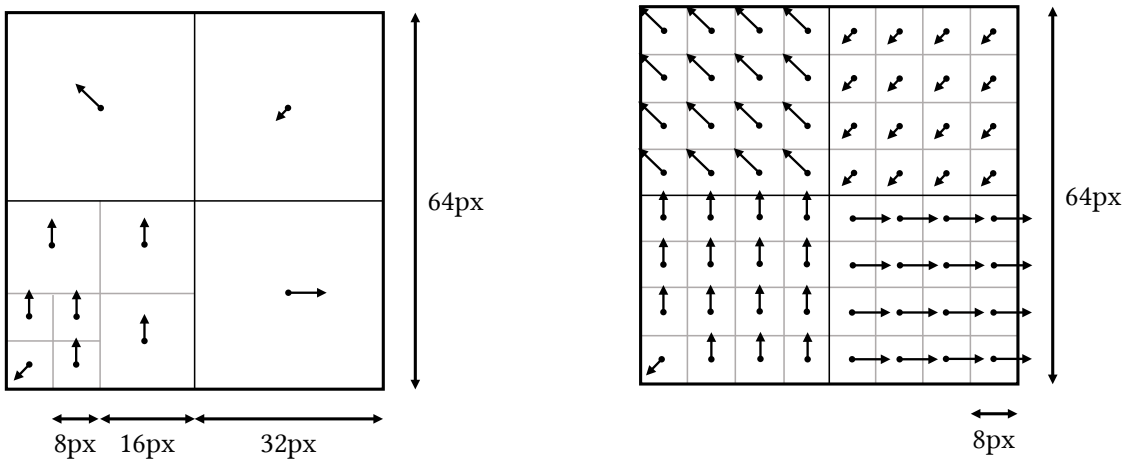
The residual requantization algorithm presented in the previous section supports requests for extremely precise bit rate targets, and it fulfills these requests in real time. This is a significant improvement over multiple coding and scalable coding, both of which require an offline transcoding step and yield video transcoded only approximately to the desired bit rate. Nonetheless, fast transcoding can be challenging to scale to many concurrent users, largely due to its high per-request computational cost.

This final section of the dissertation proposes as future work the idea to explore fully-decentralized video adaptation, in which the client is provided the necessary interface to receive the video in real time at any desired bit rate, but in which the client is responsible for performing nearly all computation. The proposed system would work by entropy decoding the source bit stream on the server to recover the syntax elements that constitute the video. These syntax elements are then exposed as-is to smart clients, through a simple HTTP API. Receivers that desire the full-quality video and have the available bandwidth are free to request all syntax elements from the provider to reconstruct the original data. Receivers that do not need the full reconstruction or that do not have the required bandwidth are able to request only the subsets of syntax elements that are most important to their application's goals. This approach offers receivers the most flexibility, as they are directly given access to individual elements of the original video stream and are able to select the elements which are the most relevant. This also represents the most distributed solution, as computation is completely deferred to receivers instead of performed by the provider.

## 7.1 Challenges

Implementing a distributed, receiver-driven video transcoder poses numerous challenges. The envisioned system could allow clients to request individual syntax elements from the source encoding. However, the syntax elements used by modern standards like H.265/HEVC are tightly coupled with each other and are often implicitly specified. For example, suppose a receiver is interested in receiving only the motion vector syntax elements for a given frame. While this sounds like a simple, reasonable request, the interdependence of symbosl in the quadtree partitioning structure complicates matters; each coding unit (CU) is a different size, and some CUs use intra-frame prediction instead of inter-frame prediction (i.e. motion vectors). A server could respond to such a request by simply listing the frame's motion vectors as they were reconstructed from the source bit stream, but unless the client already knows the quadtree structure and block-wise prediction modes for that frame, it will be unable to match the received motion vectors with their respective prediction blocks. Thus, just to *interpret* a list of motion vectors, the client is also required to request the quadtree structure and the inter/intra coding decisions for each block in the frame. Clearly, the syntax elements defined in H.265/HEVC were not designed to be independently consumed.

To solve this problem, an intermediate representation for the syntax elements must be carefully constructed, so that they can be individually received and understood by the client without needing the entire stream to be transferred as necessary context. This proposed representation will be informed by—and indeed similar to—H.265/HEVC. However, the syntax elements will be designed so that they are less interdependent. For example, to solve the motion vector problem explained above, one solution is for the server to maximally sample the motion vectors in the frame, and respond with the maximum possible number of motion vectors that could be represented in a frame (see Figure 7.1). Nonexistent motion vectors can be represented by a special symbol, so as to distinguish the case where the zero motion vector is used from the case where no motion vector is




(a) A simplified CTU quadtree depicting CU-level motion vectors

(b) The same CTU quadtree with maximally sampled CTU-level motion vectors

Figure 7.1: Clients that do not have the quadtree information for a particular CTU can still interpret motion information if it is maximally sampled.

computed. The result is that the client receives an immediately understandable representation of the motion in the frame, without requiring additional context or side information.

Crucially, this technique relies on having a pre-known, implicit syntax element addressing scheme shared by both the server and the client. For the motion vector example, the implicit addressing scheme is the address space obtained by maximally sampling the motion vectors in the frame. To ensure the client can request and interpret any piece of the H.265/HEVC stream, similar implicit address spaces must be developed for the other syntax elements used by H.265/HEVC. This will allow clients to request by address the syntax elements either individually or as block. In response, servers can extract and provide single syntax elements or a group of them.

Another significant research question is how to manage the sheer number of syntax elements specified by an H.265/HEVC video stream. To reduce the bit cost of representing these syntax elements, a standard-compliant H.265/HEVC bit stream employs two techniques: it implicitly specifies syntax elements whose value can be inferred by prior syntax elements, and it entropy codes the ones which cannot be inferred. However, an important facet of the proposed system is to unravel the interdependence between syntax elements; this means the system will not be able to take advantage of syntax element values that can be inferred. And entropy coding relies on a shared

adaptive probability model between the encoder and the decoder; this shared probability model achieves compression by adjusting the bit costs of future symbols based on prior encoded values. However, since the proposed model assumes clients are smart and independent, each client will be receiving different subsets of the source video. Thus, each client must have its own personal entropy probability model shared with the server which reflects only the data that that client has received. This means the server will have to track separate probability models for each client. Luckily, the probability model tables are quite small for H.265/HEVC, and this will require only a few bytes stored in the server session memory for each client.

Finally, the proposed model of video transfer is structured around two-way communication between the server and the client. The client requests the exact syntax elements it wants, and the server sends them, entropy coded, in the response. High data transfer levels might be a concern in this bidirectional system. One strategy for alleviating this problem may be to identify and index a few common syntax element "packages" that clients typically request, and then allow clients to subscribe to one or more of these packages by simply indicating which packages they need. This would result in a more typical one-directional information flow where the server is simply sending the client data, and the client is simply consuming it. It also has the secondary benefit of allowing the server to batch similar clients together that have the same common requests.

## 7.2 Anticipated Impact

The project outlined above, including designing and implementing the proposed system, could yield numerous practical and theoretical benefits. The system would enable real-time live video sharing with minimal server computational demand, allowing consumer-level computers to become video distribution servers that support fully scalable requests. A distributed computational approach is key to this vision, allowing most of the transcoding algorithm to be performed by the client device.

Moreover, the practicality of fully receiver-driven video distribution could be evaluated. The upward direction of consumer device capability coupled with an increase in the number of

interesting and clever AI algorithms that operate on multimedia data indicate that the time is ripe to reevaluate existing supporting infrastructure and architecture. A key aspect of this future work is to continue questioning the conventional approach by asking if a more flexible video distribution system may be preferable for certain, increasingly common use cases.

# CHAPTER 8

## Conclusion

The work that embodies this dissertation set out to explore novel video adaptation strategies that (1) preserve the initial representation as the ground truth, (2) respond gracefully to heterogeneous receiver-driven application needs, and (3) prioritize compression performance of high bandwidth requests. In pursuit of this goal, we proposed three new methods for video adaptation. First, we presented a fundamental, content agnostic entropy-based approach in which adaptation is cast as a symbol dropping problem. To evaluate the technique, we applied it to video originally coded with H.265/HEVC and video coded with M-JPEG. The results of these experiments showed that the highly predictive nature and heavy inter-dependence of H.265/HEVC syntax elements pose a challenge to the symbol dropping approach. On the other hand, adaptation with non-predictive M-JPEG yields a smooth trade-off between bit rate and reproduction quality. Furthermore, by providing a sophisticated rate controller algorithm, receivers are able to precisely target available bandwidth toward dynamically specified regions of interest within the video frame.

Next, we demonstrated a domain-specific adaptation technique called residual requantization. Residual requantization works by allowing receivers to specify a target bit rate. To achieve this target, the residual signal from the original encoding is requantized using a different QP value. A compensation mechanism is also incorporated during residual requantization to ensure that any decoder drift experienced does not propagate forward. All prediction instructions are directly reused from the source encoding, ensuring that any embedded semantic information originally captured about the scene is preserved in the adaptation. This approach is applied to H.265/HEVC coded content and compared against a state-of-the-art cascaded transcoder. The results confirm that a top-down approach yields better rate-distortion performance at the high end of the bit rate spectrum.

Finally, we proposed a speculative fully receiver-driven system in which a video's syntax elements from its initial encoding are exposed through an HTTP API. Receivers wishing to recreate the content can request the syntax elements through an expressive query language. Syntax elements can be requested individually or in groups. In the future, this approach could allow smart receivers to save bandwidth by picking and choosing exactly the syntax elements that are most relevant to their application needs.

As the world eagerly awaits the next few decades of advances in video technology, it is easy to imagine a future in which video sensors, displays, and applications continue to diversify and become more sophisticated. As these exciting new technologies begin to materialize, we must continuously reevaluate the systems on which they are built, ensuring that they are still up to the challenge of supporting the new use cases we design for them. To that end, flexibility will certainly remain a top priority for video compression, representation, and adaptation. The results presented in this dissertation are encouraging, and the techniques that were evaluated have the potential to adapt existing video content for the next generation of sophisticated, diverse systems. Ultimately, this work should be interpreted as an initial step towards a new future of more flexible, general-use video representation and adaptation strategies.

## BIBLIOGRAPHY


Ahmed, N. (1991). How i came up with the discrete cosine transform. *Digital Signal Processing*, 1(1):4–5.

Amon, P., Haoyu Li, Hutter, A., Renzi, D., and Battista, S. (2008). Scalable video coding and transcoding. In *2008 IEEE International Conference on Automation, Quality and Testing, Robotics*, volume 1, pages 336–341.

Benyaminovich, S., Hadar, O., and Kaminsky, E. (2005). Optimal transrating via dct coefficients modification and dropping. In *ITRE 2005. 3rd International Conference on Information Technology: Research and Education, 2005.*, pages 100–104. IEEE.

Bocharov, J. A. (2009). Smooth streaming technical overview. *CM-IPTV0560, Oct*, 20:18.

Boyce, J. M., Ye, Y., Chen, J., and Ramasubramonian, A. K. (2016). Overview of shvc: Scalable extensions of the high efficiency video coding standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 26(1):20–34.

Burt, P. J. and Adelson, E. H. (1987). The laplacian pyramid as a compact image code. In *Readings in computer vision*, pages 671–679. Elsevier.

Chen, J., Koc, U.-V., and Liu, K. R. (2001). *Design of digital video coding systems: a complete compressed domain approach*. CRC Press.

Cisco (2017). Cisco visual networking index: Forecast and methodology. Technical report, Cisco.

De Praeter, J., Van Wallendael, G., Slowack, J., and Lambert, P. (2017). Video encoder architecture for low-delay live-streaming events. *IEEE Transactions on Multimedia*, 19(10):2252–2266.

Deknudt, C., Corlay, P., Bacquet, A.-S., Zwingelstein-Colin, M., and Coudoux, F.-X. (2010). Reduced complexity h. 264/avc transrating based on frequency selectivity for high-definition streams. *IEEE Transactions on Consumer Electronics*, 56(4):2430–2437.

Ding, W. and Liu, B. (1996). Rate control of mpeg video coding and recording by rate-quantization modeling. *IEEE transactions on circuits and systems for video technology*, 6(1):12–20.

Fang, Z., Wang, S., Xu, S., Wu, S., Wang, Z., and Zeng, W. (2006). Multiwavelet video compression based on block motion compensation. In *Wavelet Active Media Technology And Information Processing: (In 2 Volumes)*, pages 847–853. World Scientific.

Fecheyr-Lippens, A. (2010). A review of http live streaming. *Internet Citation*, pages 1–37.

Group, M. P. E. (2017). Call for evidence on transcoding for network distributed video coding. Technical Report ISO/IEC JTC1/SC29/WG11 MPEG2017/N17058, Moving Picture Experts Group (MPEG).

Hollmann, C. and Sjöberg, R. (2018). Guided transcoding using deflation and inflation. In *Proceedings of the 23rd Packet Video Workshop*, pages 19–24.

Jdidia, S. B., Belghith, F., Jridi, M., and Masmoudi, N. (2021). A multicriteria optimization of the discrete sine transform for versatile video coding standard. *Signal, Image and Video Processing*, pages 1–9.

Jensen, A. and la Cour-Harbo, A. (2001). *Ripples in mathematics: the discrete wavelet transform*. Springer Science & Business Media.

Kekre, H., Thepade, S. D., and Maloo, A. (2010). Query by image content using color-texture features extracted from haar wavelet pyramid. *IJCA Journal Special Issue on CASCT*, pages 53–60.

Li, W. (2001). Overview of fine granularity scalability in mpeg-4 video standard. *IEEE Transactions on circuits and systems for video technology*, 11(3):301–317.

Marcellin, M. W., Gormish, M. J., Bilgin, A., and Boliek, M. P. (2000). An overview of jpeg-2000. In *Proceedings DCC 2000. Data Compression Conference*, pages 523–541. IEEE.

Mayer, C., Crysandt, H., and Ohm, J.-R. (2002). Bit plane quantization for scalable video coding. In *Visual Communications and Image Processing 2002*, volume 4671, pages 1142–1152. International Society for Optics and Photonics.

Müller, C., Lederer, S., and Timmerer, C. (2012). An evaluation of dynamic adaptive streaming over http in vehicular environments. In *Proceedings of the 4th Workshop on Mobile Video*, pages 37–42.

MulticoreWare (2019). x265.

Ohm, J.-R. (2005). Advances in scalable video coding. *Proceedings of the IEEE*, 93(1):42–56.

Ohno, Y. (2000). Cie fundamentals for color measurements. In *NIP & Digital Fabrication Conference*, pages 540–545. Society for Imaging Science and Technology.

Pedzisz, M. (2013). Beyond bt. 709. In *SMPTE 2013 Annual Technical Conference & Exhibition*, pages 1–13. SMPTE.

Peixoto, E. and Izquierdo, E. (2012). A complexity-scalable transcoder from h. 264/avc to the new hevc codec. In *2012 19th IEEE International Conference on Image Processing*, pages 737–740. IEEE.

Praeter, J. D., Hollmann, C., Sjoberg, R., Wallendael, G. V., and Lambert, P. (2021). Network-distributed video coding.

Radha, H. M., Van der Schaar, M., and Chen, Y. (2001). The mpeg-4 fine-grained scalable video coding method for multimedia streaming over ip. *IEEE Transactions on multimedia*, 3(1):53–68.

Rao, K. R., Kim, D. N., and Hwang, J. J. (2010). *Fast Fourier transform: algorithms and applications*. Springer.

Rose, K. and Regunathan, S. L. (2001). Toward optimality in scalable predictive coding. *IEEE Transactions on Image processing*, 10(7):965–976.

Rusert, T., Andersson, K., Yu, R., and Nordgren, H. (2016). Guided just-in-time transcoding for cloud-based video platforms. In *2016 IEEE International Conference on Image Processing (ICIP)*, pages 1489–1493. IEEE.

Schwarz, H., Marpe, D., and Wiegand, T. (2007). Overview of the scalable video coding extension of the h.264/avc standard. *To appear in IEEE Transactions on Circuits and Systems for Video Technology*, page 1.

Shannon, C. E. (1948). A mathematical theory of communication. *Bell system technical journal*, 27(3):379–423.

Sjöberg, R., Ducloux, X., Park, K., and Mekuria, R. (2017). Requirements for network distributed video coding (version 5). Technical Report ISO/IEC JTC1/SC29/WG11 MPEG2017/N17063, The Moving Picture Experts Group (MPEG).

Sodagar, I. (2011). The mpeg-dash standard for multimedia streaming over the internet. *IEEE MultiMedia*, 18(4):62–67.

Sole, J., Joshi, R., Nguyen, N., Ji, T., Karczewicz, M., Clare, G., Henry, F., and Duenas, A. (2012). Transform coefficient coding in hevc. *IEEE Transactions on Circuits and Systems for Video Technology*, 22(12):1765–1777.

Szedo, G. (2006). Color-space converter: Rgb to ycrcb. *Xilinx Corp*.

Tan, K. H. and Ghanbari, M. (1995). Layered image coding using the dct pyramid. *IEEE transactions on image processing*, 4(4):512–516.

Thang, T. C., Ho, Q.-D., Kang, J. W., and Pham, A. T. (2012). Adaptive streaming of audiovisual content using mpeg dash. *IEEE Transactions on Consumer Electronics*, 58(1):78–85.

Tomar, S. (2006). Converting video formats with ffmpeg. *Linux Journal*, 2006(146):10.

Van Wallendael, G., De Cock, J., and Van de Walle, R. (2012). Fast transcoding for video delivery by means of a control stream. In *2012 19th IEEE International Conference on Image Processing*, pages 733–736. IEEE.

Vetro, A., Christopoulos, C., and Sun, H. (2003). Video transcoding architectures and techniques: an overview. *IEEE Signal processing magazine*, 20(2):18–29.

Wallace, G. K. (1992). The jpeg still picture compression standard. *IEEE transactions on consumer electronics*, 38(1):xviii–xxxiv.

Wang, C.-C., Chang, Y.-S., and Huang, K.-N. (2016). Efficient coding tree unit (ctu) decision method for scalable high-efficiency video coding (shvc) encoder. In *Recent Advances in Image and Video Coding*. IntechOpen.

Wang, Q., Xiong, Z., Wu, F., and Li, S. (2002). Optimal rate allocation for progressive fine granularity scalable video coding. *IEEE Signal Processing Letters*, 9(2):33–39.

Winkler, S., Kunt, M., and van den Branden Lambrecht, C. J. (2001). Vision and video: models and applications. In *Vision Models and Applications to Image and Video Processing*, pages 201–229. Springer.

Wu, F., Li, S., and Zhang, Y.-Q. (2000). Dct-prediction based progressive fine granularity scalable coding. In *Proceedings 2000 International Conference on Image Processing (Cat. No. 00CH37101)*, volume 3, pages 556–559. IEEE.

Youn, J., Xin, J., and Sun, M.-T. (2000). Fast video transcoding architectures for networked multimedia applications. In *2000 IEEE International Symposium on Circuits and Systems (ISCAS)*, volume 4, pages 25–28. IEEE.

Yuan, H., Guo, C., Liu, J., Wang, X., and Kwong, S. (2017). Motion-homogeneous-based fast transcoding method from h. 264/avc to hevc. *IEEE Transactions on Multimedia*, 19(7):1416–1430.

Zhang, X., Li, Y., Li, J., Zhao, K., and Zhang, T. (2014). Proximate control stream assisted video transcoding for heterogeneous content delivery network. In *2014 IEEE International Conference on Image Processing (ICIP)*, pages 2552–2555. IEEE.

Zong-Yi, C., Chi-Teng, T., and Pao-Chi, C. (2013). Fast inter prediction for h. 264 to hevc transcoding. In *3rd International Conference on Multimedia Technology (ICMT-13)*, pages 1294–1301. Atlantis Press.