Accelerated Event-Based Feature Detection and Compression for Surveillance Video Systems

Andrew C. Freeman
acfreeman@cs.unc.edu
University of North Carolina
Chapel Hill, North Carolina, USA

Ketan Mayer-Patel
kmp@cs.unc.edu
University of North Carolina
Chapel Hill, North Carolina, USA

Montek Singh
montek@cs.unc.edu
University of North Carolina
Chapel Hill, North Carolina, USA

ABSTRACT

The strong temporal consistency of surveillance video enables compelling compression performance with traditional methods, but downstream vision applications operate on decoded image frames with a high data rate. Since it is not straightforward for applications to extract information on temporal redundancy from the compressed video representations, we propose a novel system which conveys temporal redundancy within a sparse decompressed representation. We leverage a video representation framework called ADΔER to transcode framed videos to sparse, asynchronous intensity samples. We introduce mechanisms for content adaptation, lossy compression, and asynchronous forms of classical vision algorithms. We evaluate our system on the VIRAT surveillance video dataset, and we show a median 43.7% speed improvement in FAST feature detection compared to OpenCV. We run the same algorithm as OpenCV, but only process pixels that receive new asynchronous events, rather than process every pixel in an image frame. Our work paves the way for upcoming neuromorphic sensors and is amenable to future applications with spiking neural networks.

CCS CONCEPTS

• **Computing methodologies** → Image compression; **Image representations**; *Scene anomaly detection*; *Image processing*; • **Applied computing** → *Surveillance mechanisms.*

KEYWORDS

event representation, event video, video processing, event vision

ACM Reference Format:
Andrew C. Freeman, Ketan Mayer-Patel, and Montek Singh. 2024. Accelerated Event-Based Feature Detection and Compression for Surveillance Video Systems. In *ACM Multimedia Systems Conference 2024 (MMSys '24), April 15–18, 2024, Bari, Italy*. ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3625468.3647618

1 INTRODUCTION

Traditionally, video systems have been built around the notion of a human viewer. Lossy compression algorithms are based on perceptual quality metrics, implying that the target application is human viewership. Increasingly, however, cameras are deployed at large scales to continuously record and analyze video with little to no human monitoring. Technology companies use such deployments to monitor their data center security, logistics companies track the flow of packages, stores detect shoplifting, transportation departments analyze traffic patterns, and intelligence agencies track suspected threats to national security.

These large-scale surveillance systems pose a number of unique issues. First, a business or government may find long-term storage costs prohibitively expensive. Although stationary cameras enable strong compression performance, 24/7 recording from hundreds or thousands of cameras can quickly saturate the operator's storage or networking budget. In critical systems, indiscriminately discarding old video data can pose a risk of safety, security, or legal liability to the operator.

Second, while traditional video codecs are highly efficient at compressing stationary surveillance video, the analysis applications are largely decoupled from the compression pipeline. Although the compressed representation may indicate that a certain region of a video is not changing over a long period of time, the compressed structure is not amenable to most applications. On the other hand, the *decompressed* representation ingested by the application is a series of standalone images with a uniform sample rate for every pixel. Therefore, vision applications may spend significant time and computational resources processing pixel values which the encoder determined to be of low salience. Meanwhile, any improvement to the computational speed of a real-time video analysis or triage pipeline can be a difference maker for human safety in emergency situations.

Recent years have seen the rapid growth of neuromorphic event-based camera sensors [20, 27]. These sensors natively capture asynchronous data samples and convey temporal stability in their raw representation [20, 27]. In this paper, we introduce several extensions to the ADΔER software framework which bridges the representational gap between event sensors, traditional framed sensors, and vision applications [19]. We evaluate the efficacy of our modifications in regard to transcoding framed surveillance video to an asynchronous representation. Our contributions are as follows:

- Introduce a simple quality parameter to automatically adjust low-level parameters of the asynchronous transcoder.
- Propose a simple, lossy compression scheme for asynchronous video with a multifaceted rate controller, intra- and inter-coding, packetized scrubbing, and context-adaptive binary arithmetic coding.
- Implement an asynchronous form of the FAST feature detector [37] and adjust the representational accuracy of pixels based on their proximity to detected features.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
MMSys '24, April 15–18, 2024, Bari, Italy
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0412-3/24/04…$15.00
https://doi.org/10.1145/3625468.3647618

- Evaluate the quality and speed of our system on the VIRAT surveillance dataset [31]. We achieve **2.5:1** compression ratios over the raw ADΔER representation and our asynchronous FAST feature detector shows a median improvement in execution speed of up to **43.7%** over OpenCV.

2 RELATED WORK

2.1 Classical Video Pipelines

The traditional video paradigm was born out of an effort to make multimedia consumption accessible to people on consumer devices. Video codecs employ techniques such as the discrete cosine transform (DCT) to induce predictable information loss according to quality directives [40]. This loss occurs only in the spatial domain, chiefly reducing the precision of slight intensity variations between pixels in close proximity to each other. Typical quality metrics include the peak signal-to-noise ratio (PSNR) [23], the structural similarity index measure (SSIM) [23], and more recently the Video Multimethod Assessment Fusion (VMAF) [26, 35]. These metrics aim to represent *perceptual* quality; that is, how a human viewer would perceive the video. To achieve better perceptual quality at lower bitrates, recent research has shifted towards the use of machine learning to optimize encoder parameters [47]. Decoded images may also be post-processed to increase the resolution [28], enhance details [25], or interpolate images at higher frame rates [10].

On the other hand, video is increasingly utilized for computer vision tasks, such as object detection, action recognition, 3D reconstruction, and scene segmentation [38]. These systems typically rely on convolutional neural networks (CNNs) to learn feature embeddings from decoded image frames. In some cases, researchers employ recurrent neural networks (RNNs) to reduce the temporal redundancy across a temporal window of input images. Yi et al. presented a framework for task-driven rate directives in video compression [45], to bridge the gap between human and machine vision.

Since modern video codecs can efficiently compress temporal redundancies, vision applications may save computational resources by processing compressed representations directly. CNNs have shown computational benefits when directly processing DCT embeddings [44], codec prediction frames and motion vectors [11, 43], and custom neural representations [6, 42]. As Wiles et al. note, existing frame-based applications and networks cannot operate on such compressed representations [42]. As such, a robust solution should be compatible with traditional applications through easily-computed image reconstructions. In a similar vein, recent work has explored *sparse convolution*, computing convolution results only on localized patches of saliency [7, 8, 22, 24, 33]. However, in all of these cases, the underlying representation is temporally synchronous. Therefore, doubling the video frame rate necessitates a doubling of the frames processed by the application, slowing the computation speed even if the motion content is similar.

2.2 Event-Based Cameras

Asynchronous sensing is a rapidly growing field for robotics research. Rather than recording image frames, these sensors capture asynchronous "events" at the precise time that a pixel has met some intensity criterion. The most common of these sensors is the Dynamic Vision System (DVS), wherein a pixel records a timestamped polarity event if its log intensity has increased or decreased by a given threshold [27]. Meanwhile, pixels whose incident intensity is static do *not* output any events. This sparse sensing scheme enables DVS cameras to achieve more than 120 dB dynamic range and microsecond temporal precision, while using very little power [20]. The Dynamic and Active Vision System (DAVIS) co-locates traditional subpixels alongside DVS pixels to simultaneously capture intensity image frames at a low rate [20].

Some vision applications have been built from the ground up with these contrast-based event sensors in mind, using spiking neural networks (SNNs) [2, 9, 12]. Many applications, however, convert the sparse event data to a frame-based representation for use with traditional processing schemes and CNNs [1, 3, 21, 29, 36, 46]. Cannici et al. and Messikommer et al. recognized that standard CNNs use many redundant computations when processing event data, and proposed sparse convolutional network architectures for DVS with applications in object detection and recognition [5, 30].

Recently, the primary manufacturer of DVS and DAVIS sensors announced the upcoming Aeveon sensor [13]. Unlike DVS, this camera natively captures floating-point intensity events. One can then reconstruct natural-looking high dynamic range images from these events, while maintaining a much higher speed and lower power usage than a traditional frame-based camera. Notably, the sensor independently tunes the exposure time of each pixel according to its incident brightness and its perceived saliency.

2.3 ADΔER

The "Address, Decimation, Δt Event Representation" (ADΔER) is our recently-introduced video representation and framework [15, 19]. It aims to be a common "narrow waist" representation for various video modalities, including framed video and DVS events. Like DVS, ADΔER is itself an event-based representation. Whereas DVS events express intensity *change*, however, ADΔER expresses absolute intensity measurements *directly*. This conceptual change makes it flexible enough to support a variety of video sources, and makes it suitable for traditional vision applications. Here we summarize the design of ADΔER and discuss its implications in our work.

Our proposed representation consists of event tuples $\langle x, y, c, D, \Delta t \rangle$. Here, x and y represent the spatial coordinates of a pixel, c is the color channel, D is the "decimation factor," and Δt is the time elapsed since the pixel last recorded an event. In the case of a monochrome video, the c component is not included in the event. Otherwise, the c index denotes the red, green, or blue channels. In a software transcoder, each pixel accumulates intensity until reaching 2^D units of intensity. The pixel luminance expressed by an event can then be calculated directly using the equation $I = \frac{2^D}{\Delta t}$.

When the transcoder input is a framed video, we have a constant parameter Δt_{ref} which defines how much time each input frame spans, measured in clock "ticks." For this paper, we set $\Delta t_{ref} = 255$ as determined to be the optimal value for 8-bit inputs [19]. We also set the time scale to match the input video with the parameter Δt_s. For example, if the input video is 30 frames per second, we

define $\Delta t_s = 255 * 30 = 7650$ ticks. These parameters are encoded as metadata in the ADΔER file header.

Inspired by prior work on an integrating event sensor [16–18], we designed our transcoder to dynamically compute the optimal D for each pixel. Under this scheme, the pixels have a contrast threshold, M, which specifies their tolerance to variations in intensity. When a pixel begins to integrate a new event, it establishes a baseline intensity, I_0. So long as incoming intensities are within the range $[I_0 - M, I_0 + M]$, the pixel will build a queue of events which maximize the D values necessary to represent the integration. With this adaptive mechanism, stable (unchanging) pixels can have a much lower event rate than dynamic (rapidly changing) pixels. Additionally, the global parameter Δt_{max} sets the maximum Δt value for any event. When Δt_{max} is reached, or when the incident intensity exceeds the contrast threshold, the pixel outputs its queue of optimal events. The Δt_{max} mechanism thus limits the maximum event latency to a user-specified amount.

We showed that ADΔER can efficiently represent data from framed video sources, with rate-distortion trade-offs driven by the contrast threshold, M [19]. Furthermore, we demonstrated state-of-the-art speed when fusing the framed and event data of DAVIS camera sources to an intensity representation, since the method does not rely on the reconstruction of hundreds or thousands of intensity frames per second; rather, the underlying representation remains asynchronous.

3 CODEC IMPROVEMENTS

In this paper, we focus on novel representations and applications for surveillance video recorded by traditional frame-based cameras. Long-form stationary video has similar motion content to extremely high-speed video, which benefits from an event-based representation (as in DVS and ADΔER). A DVS-style representation is ill-suited to surveillance video, however, since it does not represent the values of static image regions. A DAVIS-style multimodal representation rectifies that issue, but fusing events and intensity frames is computationally expensive [32, 39, 41]. Instead, we note that ADΔER shares *many* of the advantages of the upcoming Aeveon sensor technology [13], underscoring our belief that it is the inevitable direction of event-based video representation. As such, we leverage the ADΔER framework to transcode framed video to intensity events, but we introduce a number of improvements over the prior work to make it suitable for practical video systems.

3.1 Δt vs. t

Our earlier ADΔER work defined an event's temporal component, Δt, as the time elapsed since the pixel last fired an event. This scheme makes it straightforward to calculate the intensity expressed by the event, through simply calculating $\frac{2^D}{\Delta t}$. When reconstructing the intensities for playback or applications, the software tracks the running clock time for each pixel to ensure that events are correctly ordered.

This representation lends itself poorly to lossy compression, however. Suppose that, for a given pixel, we have a sequence of $\langle D, \Delta t \rangle$ events $\{e_0 = \langle 5, 100_\Delta \rangle, e_1 = \langle 5, 120_\Delta \rangle, e_2 = \langle 5, 110_\Delta \rangle\}$. If we track the running time of the pixel, we see that e_2 fires at $t = 330$ ticks. Now, suppose without loss of generality that we incur some compression loss in the Δt component of e_0, and upon reconstruction we obtain the sequence $\{e'_0 = \langle 5, 70_\Delta \rangle, e'_1 = \langle 5, 120_\Delta \rangle, e'_2 = \langle 5, 110_\Delta \rangle\}$. Then, the firing time for the e'_2 is $t = 300$ ticks. Although we incurred loss only in e_0, the change-based temporal measurement has a compounding effect on all the later events for that pixel.

To rectify this, we use an absolute t representation as the temporal component of our events. In the above example, our original sequence would be $\{e_0 = \langle 5, 100_\Sigma \rangle, e_1 = \langle 5, 220_\Sigma \rangle, e_2 = \langle 5, 330_\Sigma \rangle\}$. Then, if we incur the same loss on e_0, our reconstructed sequence would be $\{e_0 = \langle 5, 70_\Sigma \rangle, e_1 = \langle 5, 220_\Sigma \rangle, e_2 = \langle 5, 330_\Sigma \rangle\}$. During playback, we simply subtract the t component of the previous event of the pixel from the current event to obtain Δt and compute the intensity as normal. In this case, the reconstructed intensity measurements would be less accurate for e_0 (brighter) and e_1 (darker), but *not* e_2. Therefore, incurring temporal loss in one event only has a compounding effect on the reconstruction accuracy of the event immediately afterward. We utilize this absolute t representation throughout the rest of this paper, with each event being the tuple $\langle x, y, c, D, t \rangle$. We continue using the ADΔER terminology, however, since we still compute the incident intensity based on the Δt between two events.

3.2 Redefining Δt_{max}

Previously, we defined the parameter Δt_{max} as the "maximum Δt that any event can span" [19]. The reason for this definition is to ensure that a client can be guaranteed that the latest update for a pixel will be available within the Δt_{max} time span. However, there is a trade-off between pixel response time and event rate. For a completely stable pixel (with an unchanging intensity value), each halving of Δt_{max} will yield a doubling in its output event rate. If one made Δt_{max} very high, the event rate would be lower, but the client would have to wait much longer to obtain the intensity of a newly stable pixel. In a streaming setting, this behavior leads to potentially high and unpredictable latency.

To address these streaming concerns, we redefine Δt_{max} as *the maximum Δt that the first event of a newly stable pixel can span.* Suppose we were to start integrating a pixel with 1 intensity unit per tick for 768 ticks, suppose the pixel's starting D value is 8, and suppose $\Delta t_{max} = 300$. After 768 ticks, suppose that the incident intensity changes, so we must write out the pixel's event queue. Under the *prior* Δt_{max} scheme, our pixel would produce the event sequence $\{e_0 = \langle 8, 256_\Sigma \rangle, e_1 = \langle 8, 512_\Sigma \rangle, e_2 = \langle 8, 768_\Sigma \rangle\}$, where the Δt between two consecutive events is no greater than 300 ticks. Under our new Δt_{max} scheme, the pixel would produce the sequence $\{e_0 = \langle 8, 256_\Sigma \rangle, e_1 = \langle 9, 768_\Sigma \rangle\}$, where only the Δt of the first event at the baseline intensity level must be within the 300-tick threshold. We can now coalesce the remaining sequence of events into a single event with a higher D and Δt value. Therefore, we have the flexibility to set Δt_{max} lower, ensuring low pixel latency for intensity changes, without having a high event rate for stable pixels. This helps us avoid repeatedly intra-coding the same event for stable pixels in a streaming-supported compression scheme.

The new Δt_{max} scheme mitigates the event rate, making transcoding operations and applications faster, and making the representation more amenable to compression. On the other hand,

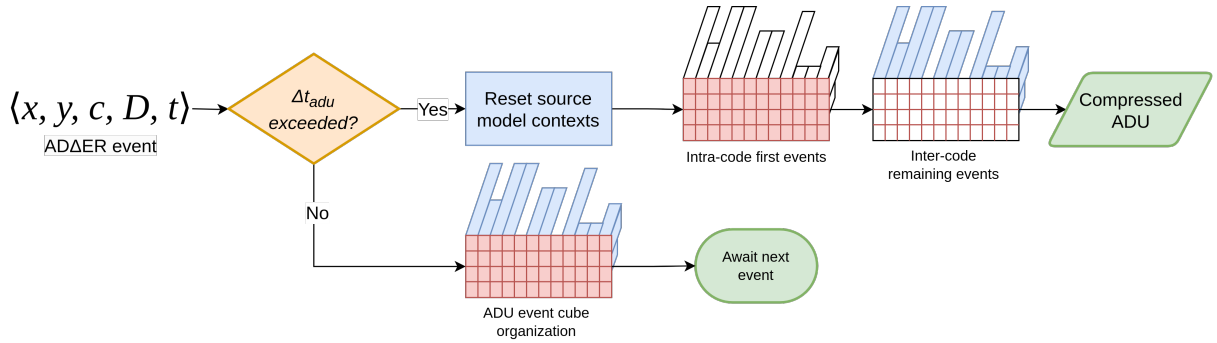


Figure 1: Simplified flowchart of our lossy compression scheme

it removes any time-bound guarantee that a client dropping into an ADΔER stream will receive an event for all pixels. That is, the client will not receive events for stable pixels until their incident intensity changes.

3.3 Adaptive Contrast Thresholds

As described in Sec. 2.3, we previously explored the use of a constant contrast threshold, M, which is the same for all pixels. To enable adaptation of content-based rate, we specify a *maximum* contrast threshold M_{max} and a threshold rate-of-change parameter, M_v. Then, a stable pixel may increase its contrast threshold by one intensity unit for every M_v intervals of Δt_{ref} time spanned, up to M_{max}. Applications within the transcoder loop may then forcibly *lower* the M of certain pixels to increase their responsiveness and accuracy as needed.

3.4 Constant Rate Factor

We found that the myriad low-level parameters available within the ADΔER system are abstruse for a general user. We sought to create a simple meta-parameter and lookup table to reasonably set the underlying variables. Taking inspiration from framed codecs such as H.264 and H.265, we call our metaparameter the constant rate factor (CRF), with values ranging from 0-9. Setting CRF to 0 yields a lossless transcoded event stream, whereas setting a high CRF value will yield greater loss but a much-reduced event rate. Specifically, the CRF table determines the parameters M, M_{max}, and M_v as described in Sec. 3.3, as well as the radius for feature-based rate adaptation described below in Sec. 5.3. We populated our lookup table such that incrementing the CRF by 1 yields a drop in PSNR reconstruction quality of 2.5-5.0 dB on a typical framed video transcode. For this work, we evaluate CRF settings 0, 3, 6, and 9, which we will refer to as `Lossless`, `High`, `Medium`, and `Low` quality settings, respectively.

3.5 Multifaceted D Control

With our modifications, we see that there are several factors which influence the D values of generated ADΔER events. We can loosely think of D as a sum of partial components

$$D = D_{intensity} + \max(D_{stability} - D_{application}, 0). \quad (1)$$

Here, $D_{intensity}$ is the baseline D-value derived from the first intensity integrated for the event. For example, if we begin integrating a pixel with 223 intensity units, then we have

$$D_{intensity} = \lfloor \log_2 223 \rfloor = 7. \quad (2)$$

$D_{stability}$ denotes the portion of D that comes from the temporal stability of a pixel's incident intensity. For example, if $M >= 3$ and we integrate 220 intensity units from three more consecutive input frames, we have

$$D_{stability} = \lfloor \log_2 223 + 220 * 3 \rfloor - D_{intensity} = 9 - 7 = 2. \quad (3)$$

The ADΔER transcoder indirectly determines $D_{stability}$ based on the contrast threshold, M, and the consistency of incoming intensities. That is, the longer a pixel integrates intensity, the higher its implied $D_{stability}$. Subsequently, a higher M makes a stable pixel more impervious to slight variations in incoming intensity, and it can integrate for a longer period of time.

Finally, $D_{application}$ is a lowering of D according to higher-level application directives. In the example above, we might set $D_{application} = 2$ to ensure that temporal variations in intensity are not averaged out. In practice, we achieve this by manually lowering the M of a pixel to make it more sensitive to variations in intensity. We note in Eq. (1) that an application cannot reduce the overall D beyond the baseline $D_{intensity}$, so that we do not unnecessarily increase the event rate.

Our new D control mechanism, in tandem with our adaptive contrast control (Sec. 3.3), gives the ADΔER framework the flexibility to allocate rate towards spatiotemporal regions of interest.

4 LOSSY COMPRESSION SCHEME

In the landscape of video representations, ADΔER is unique in allowing for two concurrent drivers of loss. In our previous work, we introduced contrast-based loss control with M, which controls the rate and distortion of raw events [19]. A single grayscale ADΔER event is 9 bytes, however, making raw files unwieldy for storage or applications. We make inroads on this data rate problem with the following custom source-modeled arithmetic coding scheme for ADΔER data. We summarize the scheme in Fig. 1.

4.1 Application Data Units

The fundamental compressed representation in our scheme is a series of Application Data Units (ADUs). We independently encode

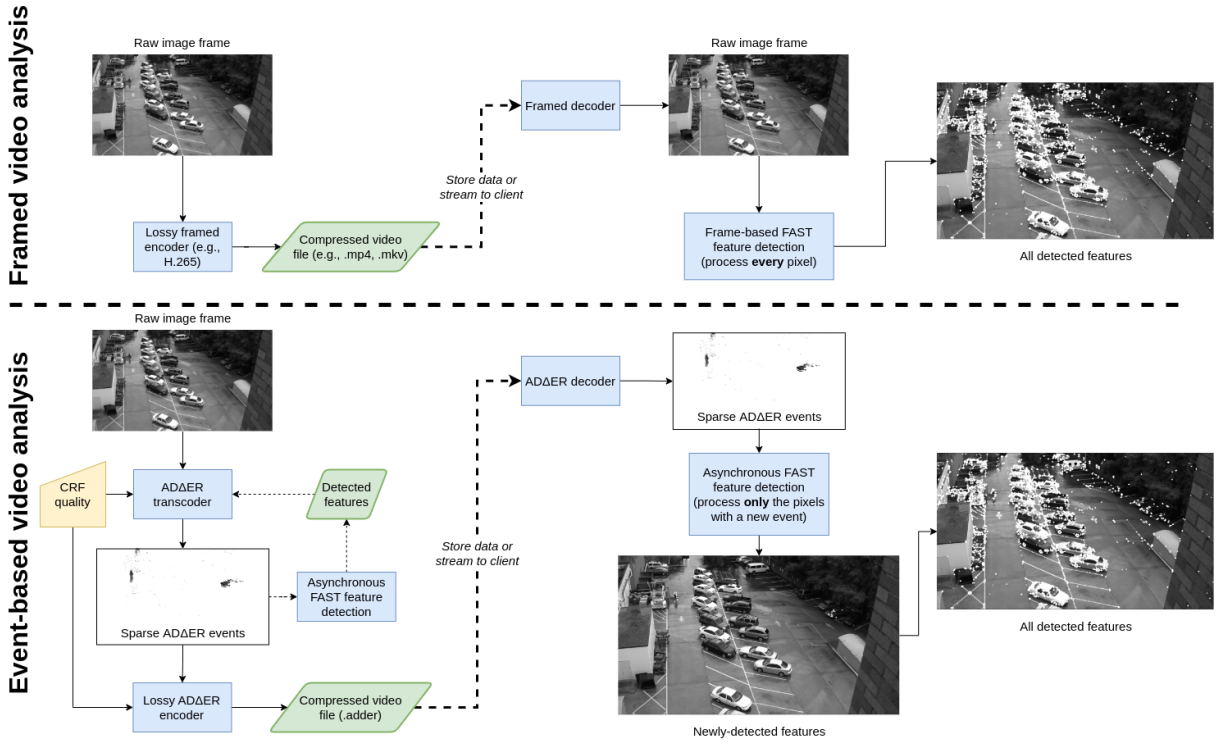


Figure 2: Comparison between FAST feature detection in a classical video system (top) and our ADΔER-based system (bottom). With ADΔER, the decompressed representation is itself sparse, meaning that the application has much less data to process.

each ADU with fresh source model contexts, to support fast scrubbing and stream drop-in. The temporal span of an ADU is denoted by Δt_{adu}. In this paper, we set $\Delta t_{adu} = \Delta t_{max}$, as defined in Sec. 3.2.

Δt_{adu} here expresses a meaning similar to the I-frame interval in framed codecs. For example, suppose we are transcoding a framed video to ADΔER with $\Delta t_{ref} = 255$ ticks and $\Delta t_{adu} = 2550$ ticks. Then, each input frame spans 255 ticks, and each ADU spans 2550 ticks. Thus, our ADU contains 10 input frames of transcoded ADΔER data. If an ADU begins at time t_0, we encode the ADU once we encounter an event with time $t' > t_0 + \Delta t_{adu}$.

4.2 Event Cubes

Within each ADU, we organize the incoming events into *event cubes*. Each event cube represents a 16×16 spatial region of pixels and a temporal range of Δt_{adu} ticks, and we maintain independent queues of events for each pixel.

We begin encoding an ADU by intra-coding the event cubes in row major order. Here, we encode only the *first* event for each pixel in each cube. Suppose that we have spatially adjacent events a and b, and we have already encoded a. We lossless-encode $D_r = D_b - D_a$ and the t residual $t_r = t_b - t_a$. If the video is in color, then our event cube contains separate pixel arrays for the red, green, and blue components, and these components are encoded in that order. We do not encode the coordinates of the events, since we have organized them spatially.

After intra-coding all the event cubes, we inter-code the remaining events for each pixel. Here, we examine temporally adjacent events a and b for a single pixel. We can leverage knowledge of the prior state of the pixel to form a t prediction, p, by

$$p_b = t'_a + \Delta t'_a \ll D_r, \tag{4}$$

where t'_a is the reconstructed timestamp of the pixel's last event, $\Delta t'_a$ is the reconstructed Δt for the pixel's last event, and D_r is the D residual. We then determine how much loss we can apply to the t prediction residual, $t_r = t_b - p_b$. For this, we iteratively right-shift the bits of the t prediction residual and calculate the intensity, I', that the decoder would obtain when reconstructing the t given the shifted residual and the shift amount, s. Our equation for s is

$$\underset{s}{\operatorname{argmax}} \left(I' = \frac{2^D}{(r \ll s) + p_b - t_0} : I - M_{max} < I' < I + M_{max} \right), \tag{5}$$

where I is the original intensity and M_{max} is our maximum contrast threshold as described in Sec. 3.3. Without the M_{max} limitation, a large prediction residual is likely to create salt-and-pepper noise when it is bit shifted. We encode D_r, s, and t_r for each event remaining in a pixel's event queue before proceeding to the next pixel in row-major order.

4.3 CABAC

We use a context-adaptive binary arithmetic coder (CABAC) [34] to perform entropy coding on our ADU data structures. We use separate contexts for the D residuals, t residuals, and s (bit shifts). We reserve a symbol in the D residual context to denote when the

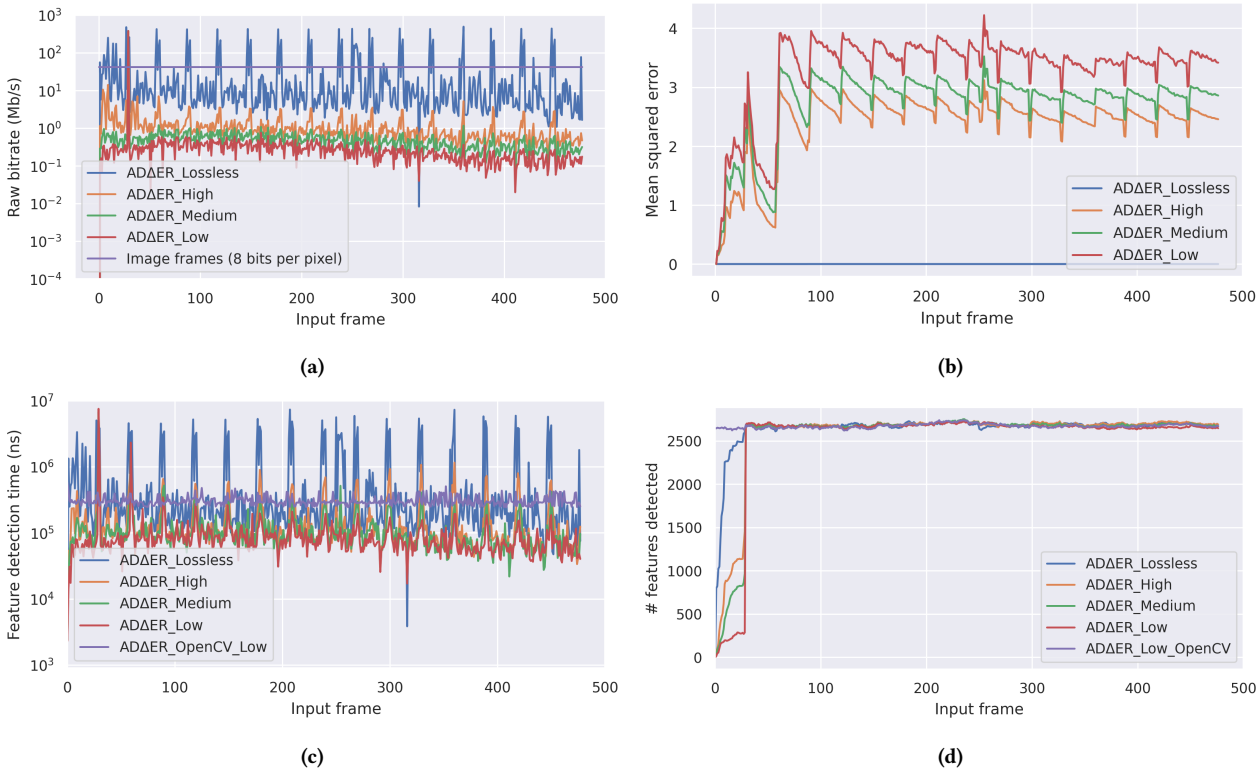


Figure 3: Key metrics gathered for a particular video. The `Lossless` lines are achievable in the work of [19] with $M = 0$, while the other lines result from our new CRF mechanism (Sec. 3.4). (a) The bitrates of the raw ADΔER representations (before arithmetic coding) at our four quality levels. For comparison, the bitrate of a raw decoded image frame is constant. (b) The mean squared error of framed reconstructions of the raw ADΔER events. (c) The execution time for FAST feature detection. (d) The total number of detected features present, over time.

decoder must move to a different spatial unit. We variously employ this symbol to indicate that an event cube does not contain any events (similar to "skip" blocks in H.265 [40]), that an individual pixel does not contain any events, and that we have completed encoding all the events for a pixel. When all the events in an ADU have been compressed, we encode a reserved "end of sequence" symbol and reset the CABAC state. In this way, we support stream scrubbing and drop in, with granularity matching the ADU interval.

5 FEATURE DETECTION

A driving motivation for our work is to enable the development of faster video analysis applications and content-based rate adaptation. To explore the utility of ADΔER in an end-to-end system, we adapted the FAST feature detection algorithm for the asynchronous paradigm [37]. The FAST feature detector examines pixels that lie along a circle around a given candidate pixel. For a given streak size n (e.g., 9 pixels), we say that the candidate pixel is a *feature* if at least n contiguous pixels in the circle exceed the candidate pixel's value plus or minus some pre-determined threshold.

5.1 Asynchronous Operation

In a traditional video analysis pipeline, the input to an application such as the FAST detector is an entire decompressed image frame, as illustrated in Fig. 2. Then, the feature detector iterates through every pixel in the image and tests it as a candidate feature. In a video context, we may visit and test many pixels which have not changed since the previous image frame. One could first calculate the pixelwise difference between the current and previous frames, and run feature detection only on the pixels that have changed, but this operation comes with its own computational costs on the decoder end.

With ADΔER, by contrast, *the decompressed representation is already sparse*. Our application layer can simply keep a single reconstructed intensity image in memory, and update an individual pixel value for each new event that it receives. When the application ingests a new event, the feature detector may test that *one pixel*, rather than all the pixels in an image.

5.2 Implementation Details

We used the OpenCV [4] implementation of the FAST feature detector as our reference. The algorithm includes an iterator loop through all pixels in an image, so we adapted only the interior



(a) H.265-compressed input
(b) Reconstructed intensities
(c) Event D values
(d) Δt between events
(e) Reconstructed intensities
(f) Event D values
(g) Δt between events

Figure 4: Zoomed-in view of the effect of feature-driven rate adaptation during transcode. (a) shows the input to the transcoder, which was compressed with H.265 at CRF level 23. (b)-(d) show views of the events transcoded under the `Low` quality setting. (e)-(g) show views of the same transcode setting and feature-driven rate adaptation enabled, as described in Sec. 5.3. The D and Δt images are normalized, such that darker pixels correspond to smaller D and Δt, respectively.

portion to apply the operation to a single pixel with an index argument. We ported the algorithm to Rust for interoperability with the rest of the ADΔER codebase. For this paper, we did not explore an asynchronous implementation of non-maximal feature suppression, which is an optional filtering step in the OpenCV implementation. To verify that the choice of programming language by itself does not produce a performance gain, we tested the speed of our Rust algorithm in processing entire image frames, synchronously. We found that our synchronous Rust algorithm is 6-10 times *slower than* its OpenCV counterpart (written in C++), owing to the high optimization level that the OpenCV project has achieved.

5.3 Feature-Driven Rate Control

We can achieve significant compression of the raw format by having high contrast thresholds while *transcoding* a video to ADΔER, as we discussed with our CRF parameter in Sec. 3.4. Conversely, we may allocate more bandwidth to regions of high salience by *lowering* the contrast thresholds for individual pixels in those regions. As illustrated in Fig. 2, we can execute an application such as our asynchronous feature detector not only during video playback, but also during the transcoding process. For the latter case, we devised a straightforward scheme which throttles down the contrast threshold, M, of all pixels within a certain distance to a newly detected feature. This distance is determined by the lookup table for the global CRF parameter (Sec. 3.4), where higher CRF values correspond to smaller adjustment radii. Since thresholds increase over time, as described in Sec. 3.3, these pixels will gradually lower their event rate if their intensities are stable. By this, we may easily choose to prioritize high quality in the spatiotemporal regions known to be of application-level importance. Fig. 4 shows a `Low` quality transcode with and without feature-driven rate adaptation enabled.

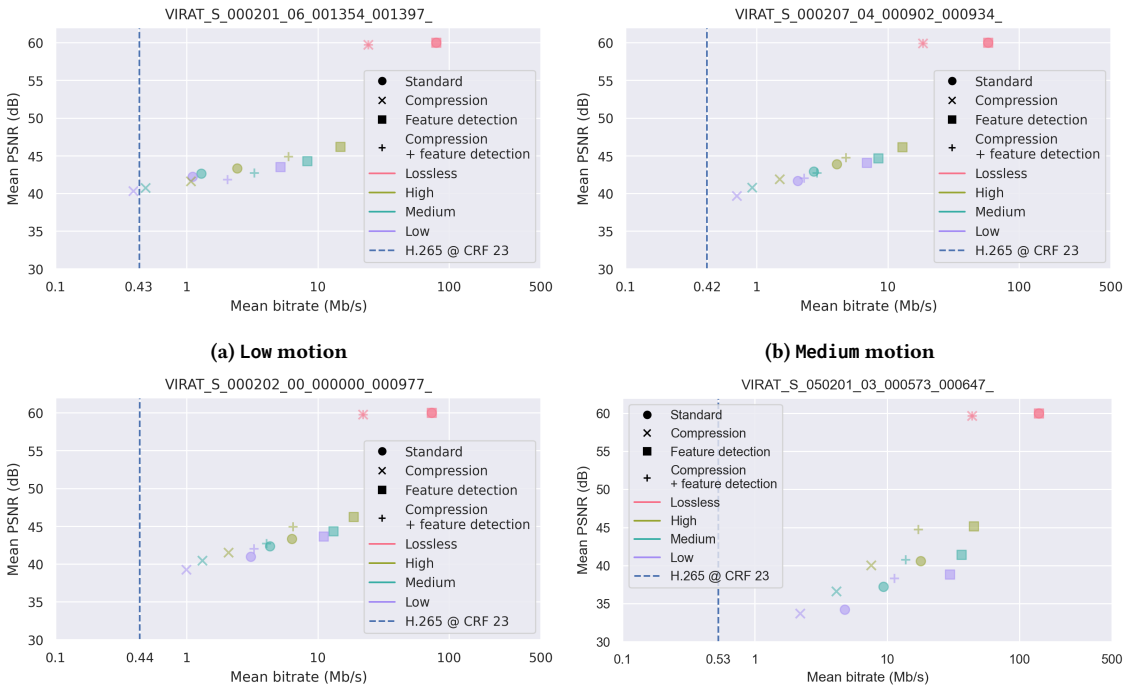
6 EVALUATION

As this work is a novel method for representing, compressing, and processing surveillance video, we present a new evaluation scheme on an existing video dataset.

6.1 Dataset and Experiments

We utilized the VIRAT video surveillance dataset [31]. This dataset contains sequences from stationary cameras recording the movements of people and vehicles in outdoor public spaces. We randomly sampled 132 videos from the dataset and re-encoded them for greater throughput and data diversity with our experiment. We scaled each video to 640×360 resolution, converted it to single-channel grayscale, and encoded the first 480 frames in H.265 with FFmpeg [14]. We instructed FFmpeg to use CRF value 23 for moderate loss and have an I-frame interval of 30.

We fed these H.265-encoded videos as input to the ADΔER transcoder. We transcoded each video at ADΔER CRF `Lossless`,



(a) `Low` motion **(b) `Medium` motion**

(c) `High` motion **(d) `Very High` motion**

Figure 5: Representative rate-distortion curves showing the effect of our transcoder quality settings and feature-driven rate adaptation. At the `Low` quality setting, our compressed representation approaches or surpasses the bitrate of the H.265-encoded source video, while maintaining a high PSNR value. Note that the H.265 data point for each video expresses only the bitrate, since its PSNR (with reference to itself) is undefined. We show one video per plot for readability, since the other videos in each motion category follow similar patterns.

`High`, `Medium`, and `Low` settings. We ran each transcode level both with and without the feature-driven rate control mechanism described in Sec. 5.3, for a total of eight experiments on each video. In all cases, we set $\Delta t_{ref} = 255$ ticks and $\Delta t_{adu} = \Delta t_{max} = 7650$ ticks. That is, each compressed ADΔER ADU spans $7650/255 = 30$ input frames, matching the I-frame interval of the source videos. Notable metrics we collected were feature detection speed, framed reconstruction quality before and after source-modeled arithmetic coding, and compression performance. We ran all experiments on an AMD Ryzen 2700X CPU with 8 cores and 16 threads. The OpenCV implementation of FAST feature detection is single-threaded, so we executed that portion of our system on a single thread for the sake of fair comparison.

6.2 Results

We illustrate various metrics for a single video in Fig. 3. We found that the periodic spikes apparent in Figs. 3a to 3c are due to the I-frame interval of the H.265-encoded source. Large quality changes occur in the source encoding every 30 input frames, and this leads to a corresponding jump in ADΔER data rate, especially at higher quality levels. Fig. 3d illustrates that the feature detector may take up to Δt_{max} ticks to detect the first instance of features located in stable pixel regions. In Figs. 3c and 3d, we compare the performance of running the frame-based OpenCV feature detector on a reconstruction of the raw ADΔER stream at `Low` quality. For visualization purposes, we do not show the OpenCV results at other quality levels, but the results were virtually identical. The key point is that the frame-based application speed is nearly constant, and does not adapt to the underlying video content, whereas the ADΔER implementation speed can vary widely depending on the number of events.

Through qualitative examination, we found that the ADΔER compression performance is highly dependent on the amount of motion in a video. We divide the videos into four motion categories, based on the difference between the H.265 bitrate and the compressed bitrate of the `Low` ADΔER quality transcode without feature detection. These motion categories are `Low` (lower bitrate than H.265), `Medium` (within 1×-2× the H.265 bitrate), `High` (within 2×-3× the H.265 bitrate), and `Very High` (3× the H.265 bitrate or greater). Of the 132 videos in our dataset, this division placed 7 videos in the `Low` motion class, 67 in `Medium` motion, 25 in `High` motion, and 33 in `Very High` motion. The `Very High` motion videos tended to show moving people or vehicles close to the camera, or high wind activity causing the camera and foliage to move substantially.

For the majority of videos tested at the `Low` quality setting, the ADΔER compression performance is less than 2× that of the H.265-encoded source video, while maintaining a high PSNR value. This

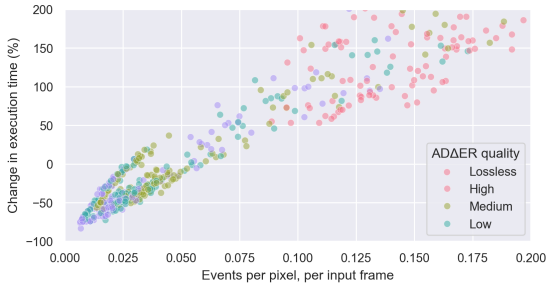


Figure 6: The effect of event rate on FAST feature detection speed, compared to the frame-based OpenCV implementation. If there is less than 1 ADΔER event for every 40 pixels, the asynchronous FAST detector is faster.

	Quality \ Motion	Low	Medium	High	Very High	All
		Median reconstructed PSNR (dB)				
Normal	High	40.2	40.9	40.7	39.2	40.5
Normal	Medium	38.8	39.4	39.5	35.4	38.8
Normal	Low	37.8	37.9	38.4	32.2	37.3
FAST	High	44.0	44.7	44.3	45.2	44.6
FAST	Medium	42.3	42.3	41.6	41.4	42.0
FAST	Low	41.4	41.3	40.7	37.9	40.9

Table 1: Median PSNR of the compressed ADΔER videos compared to the input H.265 video. We reconstructed framed videos from the ADΔER representations to evaluate the PSNR. The columns indicate the motion category, while the rows indicate the ADΔER transcoder quality under both the standard method and with rate adaptation based on FAST feature detection. The use of our feature-driven adaptation mechanism described in Sec. 5.3 increases the quality.

Quality \ Motion	Low	Medium	High	Very High	All
	Median change in feature detection time (%)				
Lossless	150.8	125.3	190.3	444.5	163.3
High	-41.0	-35.1	-13.2	99.4	-16.6
Medium	-60.4	-58.7	-30.9	81.5	-33.3
Low	**-67.5**	-56.7	-38.4	53.0	-43.7

Table 2: Median change in FAST feature detection time between the frame-based OpenCV implementation and our asynchronous implementation. The columns indicate the motion category, while the rows indicate the ADΔER transcoder quality. Results less than 0 indicate that our method performed faster than OpenCV.

	Quality \ Motion	Low	Medium	High	Very High	All
		Median change in bitrate, H.265 to ADΔER (%)				
Normal	Lossless	5330.3	4717.2	5916.1	6491.2	5443.6
Normal	High	181.2	335.4	411.3	1210.4	391.7
Normal	Medium	15.3	130.2	213.5	635.8	171.6
Normal	Low	**-9.9**	51.9	292.4	214.3	84.6
FAST	Lossless	5330.3	4717.2	5916.1	6491.2	5443.6
FAST	High	2354.4	1279.9	1423.3	2964.2	1384.7
FAST	Medium	768.6	848.6	929.8	2412.2	943.2
FAST	Low	601.9	646.3	702.6	2005.3	739.0

Table 3: Median change in compressed video size between the H.265 video source and our transcoded ADΔER representation. The columns indicate the motion category, while the rows indicate the ADΔER transcoder quality under both the standard method and with rate adaptation based on FAST feature detection. Results less than 0 indicate that our compressed ADΔER representation has a lower bitrate than the H.265 source.

result is significant since our compression scheme is naive compared to the advanced techniques of modern frame-based codecs. Specifically, our current scheme does not employ variable block sizes, motion compensation, or frequency transforms. Thus, we expose the inefficiency of frame-based methods for compressing video with high temporal redundancy, and we expect to handily surpass these codecs with future development of our compression scheme. While the feature-driven rate adaptation improves the overall PSNR, as shown in Tab. 1, we note that the quality improvement is by design centered around regions with moving features. We visualize this in Fig. 4, showing the effect of feature detection on reconstruction quality. When features are detected on the moving vehicles, the transcoder makes nearby pixels more sensitive, so that Fig. 4e avoids the temporal smoothing artifacts present in Fig. 4b. However, we see that the pixels far from the moving bus (where new features were detected) maintain a lower quality due to temporal averaging, and artifacts in those regions are still visible in Fig. 4e.

Fig. 5 plots the rate-distortion curves for a representative video in each of our motion classes. We distinguish the four ADΔER quality levels by color. We mark the bitrate of the H.265-encoded source video with a dashed line. The PSNR for each marker is calculated in reference to this source video. We limit our maximal PSNR to 60 dB for visualization, since the Lossless PSNR was effectively infinite. The circular markers show the performance of transcoding to ADΔER without feature-driven rate adaption or source-modeled compression, whereas the x-shaped markers denote the performance of applying compression without feature-driven rate adaption. We see that as the ADΔER quality decreases, the bitrate and PSNR uniformly decrease as well. At any lossy quality level, we can see that applying our compression scheme yields a substantial drop in bitrate and a small reduction in PSNR. In fact, we found that our source-modeled compression scheme achieves 2.5:1 compression ratios at all quality levels and a reduction in PSNR of just 1.2-1.6 dB. Furthermore, this trend holds true for all four motion classes. The square markers denote the results with feature-driven rate adaption enabled, and the plus-sign markers denote feature-driven rate adaption and source-modeled compression. Examining these two sets of data points, we see the same trends for compression performance.

However, the overall data rates with feature detection are higher than the standard transcode results at the same ADΔER quality level, as we increase the pixel sensitivities (and thus the event rate) near detected features. Furthermore, the `Lossless` quality exhibits extremely high data rates. Enabling feature-driven rate adaptation does not increase the bitrate at the `Lossless` level, since all pixels are already at their maximum sensitivity.

Notably, we see that the `Very High` motion videos have a greater reduction in PSNR as the ADΔER quality decreases, as shown in Fig. 5d. However, this motion category also shows a greater increase in PSNR when enabling feature-driven rate adaption. This is evident in the relationship between the circular and square markers with the same color in Fig. 5d, and a higher median increase in PSNR compared to the other motion classes in Tab. 1.

The time to encode an ADU as described in Sec. 6.1 at 360p resolution and `High` ADΔER quality on a single CPU thread averages 216 ms, while the decode speed averages 123 ms. Thus, excluding the cost of transcoding the input frames to ADΔER, we can compress up to 138 input frames per second in real time. By contrast, prior compression work on a precursor to ADΔER was only computed offline due to extremely slow performance [18, 19].

Furthermore, we show the relationship between the raw ADΔER event rate and the speed of asynchronous feature detection in Fig. 6. We see that if the decoded event rate is less than 1 event for every 40 pixels in the image plane, event-based feature detection on the sparse events executes faster than frame-based feature detection on all the pixel intensities. With surveillance video sources, if we allow a slight amount of temporal loss with a non-zero M, our transcoded representation *easily* achieves this sparse event rate. We detail these results at our various quality levels and motion classes in Tab. 2. Across our 132 videos, the median asynchronous feature detection speed is faster than the frame-based method for all the quality levels except `Lossless` (which produces a very high data rate). At the `Low` quality, we see an overall **43.7% speed improvement** over the frame-based OpenCV implementation. We see the best results for the `Low` motion class, where at `Low` quality our median feature detection speed is **nearly two-thirds faster** than that of OpenCV. As we noted in Sec. 5.2, any speed improvement here is due to the efficiency of the sparse representation, *not* our particular FAST implementation.

Meanwhile, the `Very High` motion class produces substantially more ADΔER events (Fig. 5d), and has slower feature detection performance than OpenCV. As noted above, the camera placement and wind-induced camera motion of these videos is atypical within the VIRAT dataset, and we would not expect such results in commercial surveillance camera deployments. Even still, we expect that incorporating motion compensation in our source-modeled encoder will greatly improve the compression performance of such videos. Since our results show that the event-based application speed depends on the *decompressed* data rate, however, a robust ADΔER motion compensation scheme for high-motion video should not merely improve the prediction accuracy of the encoder; rather, it should reduce the raw event rate itself.

Finally, Tab. 3 shows the median percentage change in bitrate from the H.265 source to our compressed ADΔER representations. We see that our FAST-driven rate adaptation greatly increases the overall bitrate. Since the mechanism concentrates the higher event rate near salient regions, however, this underscores the claim that a more robust event prediction mechanism will help our encoder performance. On the other hand, we see that in scenes with low motion, transcoded at low quality, we outperform H.265 (up to 9.9%), with only a minor drop in PNSR (Tab. 1).

Despite our naive compression scheme, these results are extremely promising: in scenes with high temporal redundancy, we can achieve higher compression ratios than standard video codecs *and* faster speed than standard applications. Our decompressed bitrate is concentrated near the beginning of a video, but can drop to near-zero during periods of little motion (Fig. 3).

7 FUTURE WORK

We note that any convolution kernel or iterative image processing algorithm can easily be adapted to operate asynchronously on sparse ADΔER events. In the future, we will explore convolutional algorithms such as edge detection, sharpening, and blur filters, and develop spiking neural networks for fully event-based applications. Additionally, we will work to improve the sophistication of our source-modeled compression with variable cube sizes, motion compensation, and better prediction schemes. Finally, we will work to optimize the performance of our system on event camera data sources and non-surveillance framed video. To make such systems practical for real-time performance at high resolutions, we will work to develop a hardware transcoder with a Field Programmable Gate Array (FPGA).

8 CONCLUSION

This paper proposes a number of extensions to the ADΔER video framework to enable simple quality control, application-driven rate adaptation of the raw representation, and robust source-modeled arithmetic coding. Overall, we find a unique set of trade-offs between video quality, application speed, and application accuracy in this asynchronous paradigm. Since vision applications typically operate on decoded intensity representations, the ADΔER representation makes possible application acceleration if the decoded data rate is sufficiently lower than that of a frame-based system. We can achieve lower ADΔER rates by reducing the transcoder quality, but such a change can potentially harm the accuracy of downstream applications. We show that our FAST feature detection can be incorporated into the compression loop itself, to ensure that high quality is preserved in the regions likely to be of highest salience for other downstream applications, while enabling faster application speed. In large-scale video surveillance system deployments, even a modest reduction in application computation time can translate to hundreds of thousands of dollars in savings per year. As native event-based intensity sensors such as Aeveon [13] enter the fold, our work shows a robust system to intelligently reduce high event rates and adapt vision algorithms to the asynchronous paradigm.

ACKNOWLEDGMENTS

This work is partially supported by a grant from the United States Department of Defense.

REFERENCES

[1] Raymond Baldwin, Ruixu Liu, Mohammed Mutlaq Almatrafi, Vijayan K Asari, and Keigo Hirakawa. 2022. Time-Ordered Recent Event (TORE) Volumes for Event Cameras. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2022), 1–1. https://doi.org/10.1109/TPAMI.2022.3172212

[2] Thomas Barbier, Celine Teuliere, and Jochen Triesch. 2021. Spike Timing-Based Unsupervised Learning of Orientation, Disparity, and Motion Representations in a Spiking Neural Network. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*. 1377–1386.

[3] Ryad Benosman, Charles Clercq, Xavier Lagorce, Sio-Hoi Ieng, and Chiara Bartolozzi. 2014. Event-Based Visual Flow. *IEEE Transactions on Neural Networks and Learning Systems* 25, 2 (2014), 407–417. https://doi.org/10.1109/TNNLS.2013.2273537

[4] G. Bradski. 2000. The OpenCV Library. *Dr. Dobb's Journal of Software Tools* (2000).

[5] Marco Cannici, Marco Ciccone, Andrea Romanoni, and Matteo Matteucci. 2019. Asynchronous Convolutional Networks for Object Detection in Neuromorphic Cameras. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. 1656–1665. https://doi.org/10.1109/CVPRW.2019.00209

[6] Hao Chen, Bo He, Hanyu Wang, Yixuan Ren, Ser-Nam Lim, and Abhinav Shrivastava. 2021. NeRV: Neural Representations for Videos. In *Advances in Neural Information Processing Systems*, A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan (Eds.). https://openreview.net/forum?id=BbikqBWZTGB

[7] Y. Chen, Y. Li, X. Zhang, J. Sun, and J. Jia. 2022. Focal Sparse Convolutional Networks for 3D Object Detection. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, Los Alamitos, CA, USA, 5418–5427. https://doi.org/10.1109/CVPR52688.2022.00535

[8] Yi-Chen Chen, Vishal M. Patel, Sumit Shekhar, Rama Chellappa, and P. Jonathon Phillips. 2013. Video-based face recognition via joint sparse representation. In *2013 10th IEEE International Conference and Workshops on Automatic Face and Gesture Recognition (FG)*. 1–8. https://doi.org/10.1109/FG.2013.6553787

[9] Georgi Dikov, Mohsen Firouzi, Florian Röhrbein, Jörg Conradt, and Christoph Richter. 2017. Spiking Cooperative Stereo-Matching at 2 ms Latency with Neuromorphic Hardware. In *Biomimetic and Biohybrid Systems*, Michael Mangan, Mark Cutkosky, Anna Mura, Paul F.M.J. Verschure, Tony Prescott, and Nathan Lepora (Eds.). Springer International Publishing, Cham, 119–137.

[10] Jiong Dong, Kaoru Ota, and Mianxiong Dong. 2023. Video Frame Interpolation: A Comprehensive Survey. *ACM Trans. Multimedia Comput. Commun. Appl.* 19, 2s, Article 78 (may 2023), 31 pages. https://doi.org/10.1145/3556544

[11] Samuel Felipe dos Santos, Nicu Sebe, and Jurandy Almeida. 2019. CV-C3D: Action Recognition on Compressed Videos with Convolutional 3D Networks. In *2019 32nd SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI)*. 24–30. https://doi.org/10.1109/SIBGRAPI.2019.00012

[12] Hadar Cohen Duwek, Albert Shalumov, and Elishai Ezra Tsur. 2021. Image Reconstruction From Neuromorphic Event Cameras Using Laplacian-Prediction and Poisson Integration With Spiking and Artificial Neural Networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*. 1333–1341.

[13] Kynan Eng. 2023. *Kynan Eng at CVPR 2023 Workshop on Event-based Vision*. Youtube. https://www.youtube.com/watch?v=tv-GqKg4Mak&ab_channel=RPGWorkshops

[14] FFmpeg Project. 2021. *FFmpeg*. https://ffmpeg.org/

[15] Andrew C. Freeman. 2023. The ADER Framework: Tools for Event Video Representations. In *Proceedings of the 14th Conference on ACM Multimedia Systems, MMSys 2023, Vancouver, BC, Canada, June 7-10, 2023*. ACM, 343–347. https://doi.org/10.1145/3587819.3593028

[16] Andrew C. Freeman, Chris Burgess, and Ketan Mayer-Patel. 2021. Motion Segmentation and Tracking for Integrating Event Cameras. In *Proceedings of the 12th ACM Multimedia Systems Conference* (Istanbul, Turkey) *(MMSys '21)*. Association for Computing Machinery, New York, NY, USA, 1–11. https://doi.org/10.1145/3458305.3463373

[17] Andrew C. Freeman and Ketan Mayer-Patel. 2020. Integrating Event Camera Sensor Emulator. In *Proceedings of the 28th ACM International Conference on Multimedia* (Seattle, WA, USA) *(MM '20)*. Association for Computing Machinery, New York, NY, USA, 4503–4505. https://doi.org/10.1145/3394171.3414394

[18] Andrew C. Freeman and Ketan Mayer-Patel. 2021. Lossy Compression for Integrating Event Cameras. In *2021 Data Compression Conference (DCC)*. 53–62. https://doi.org/10.1109/DCC50243.2021.00013

[19] Andrew C. Freeman, Montek Singh, and Ketan Mayer-Patel. 2023. An Asynchronous Intensity Representation for Framed and Event Video Sources. In *Proceedings of the 14th ACM Multimedia Systems Conference* (Vancouver, BC, Canada) *(MMSys '23)*. Association for Computing Machinery, New York, NY, USA, 1–12. https://doi.org/10.1145/3587819.3590969

[20] G. Gallego, T. Delbruck, G. M. Orchard, C. Bartolozzi, B. Taba, A. Censi, S. Leutenegger, A. Davison, J. Conradt, K. Daniilidis, and D. Scaramuzza. 2020. Event-based Vision: A Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2020), 1–1. https://doi.org/10.1109/TPAMI.2020.3008413

[21] Daniel Gehrig, Antonio Loquercio, Konstantinos G. Derpanis, and Davide Scaramuzza. 2019. End-to-End Learning of Representations for Asynchronous Event-Based Data. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*.

[22] Benjamin Graham, Martin Engelcke, and Laurens van der Maaten. 2018. 3D Semantic Segmentation with Submanifold Sparse Convolutional Networks. *CVPR* (2018).

[23] Alain Horé and Djemel Ziou. 2010. Image Quality Metrics: PSNR vs. SSIM. In *2010 20th International Conference on Pattern Recognition*. 2366–2369. https://doi.org/10.1109/ICPR.2010.579

[24] Mikael Jacquemont., Luca Antiga., Thomas Vuillaume., Giorgia Silvestri., Alexandre Benoit., Patrick Lambert., and Gilles Maurin. 2019. Indexed Operations for Non-rectangular Lattices Applied to Convolutional Neural Networks. In *Proceedings of the 14th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP 2019) - Volume 5: VISAPP*. INSTICC, SciTePress, 362–371. https://doi.org/10.5220/0007364303620371

[25] Chongyi Li, Chunle Guo, Linghao Han, Jun Jiang, Ming-Ming Cheng, Jinwei Gu, and Chen Change Loy. 2022. Low-Light Image and Video Enhancement Using Deep Learning: A Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44, 12 (2022), 9396–9416. https://doi.org/10.1109/TPAMI.2021.3126387

[26] Zhi Li, Anne Aaron, Ioannis Katsavounidis, Anush Moorthy, and Megha Manohara. 2016. Toward a practical perceptual video quality metric. *The Netflix Tech Blog* 6, 2 (2016).

[27] P. Lichtsteiner, C. Posch, and T. Delbruck. 2006. A 128 X 128 120db 30mw asynchronous vision sensor that responds to relative intensity change. In *2006 IEEE International Solid State Circuits Conference - Digest of Technical Papers*. 2060–2069.

[28] Hongying Liu, Zhubo Ruan, Peng Zhao, Chao Dong, Fanhua Shang, Yuanyuan Liu, Linlin Yang, and Radu Timofte. 2022. Video Super-Resolution Based on Deep Learning: A Comprehensive Survey. *Artif. Intell. Rev.* 55, 8 (dec 2022), 5981–6035. https://doi.org/10.1007/s10462-022-10147-y

[29] Ana Maqueda, Antonio Loquercio, Guillermo Gallego, Narciso Garcia, and Davide Scaramuzza. 2018. Event-Based Vision Meets Deep Learning on Steering Prediction for Self-Driving Cars. 5419–5427. https://doi.org/10.1109/CVPR.2018.00568

[30] Nico Messikommer, Daniel Gehrig, Antonio Loquercio, and Davide Scaramuzza. 2020. Event-based Asynchronous Sparse Convolutional Networks. *European Conference on Computer Vision. (ECCV)*. http://rpg.ifi.uzh.ch/docs/ECCV20_Messikommer.pdf

[31] Sangmin Oh, Anthony Hoogs, Amitha Perera, Naresh Cuntoor, Chia-Chih Chen, Jong Taek Lee, Saurajit Mukherjee, J. K. Aggarwal, Hyungtae Lee, Larry Davis, Eran Swears, Xioyang Wang, Qiang Ji, Kishore Reddy, Mubarak Shah, Carl Vondrick, Hamed Pirsiavash, Deva Ramanan, Jenny Yuen, Antonio Torralba, Bi Song, Anesco Fong, Amit Roy-Chowdhury, and Mita Desai. 2011. A large-scale benchmark dataset for event recognition in surveillance video. In *CVPR 2011*. 3153–3160. https://doi.org/10.1109/CVPR.2011.5995586

[32] Liyuan Pan, Cedric Scheerlinck, Xin Yu, Richard Hartley, Miaomiao Liu, and Yuchao Dai. 2019. Bringing a Blurry Frame Alive at High Frame-Rate With an Event Camera. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

[33] Mathias Parger, Chengcheng Tang, Christopher D. Twigg, Cem Keskin, Robert Wang, and Markus Steinberger. 2022. DeltaCNN: End-to-End CNN Inference of Sparse Frame Differences in Videos. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 12487–12496. https://doi.org/10.1109/CVPR52688.2022.01217

[34] W. B. Pennebaker, J. L. Mitchell, G. G. Langdon, and R. B. Arps. 1988. An overview of the basic principles of the Q-Coder adaptive binary arithmetic coder. *IBM Journal of Research and Development* 32, 6 (1988), 717–726. https://doi.org/10.1147/rd.326.0717

[35] Reza Rassool. 2017. VMAF reproducibility: Validating a perceptual practical video quality metric. In *2017 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB)*. 1–2. https://doi.org/10.1109/BMSB.2017.7986143

[36] Henri Rebecq, Timo Horstschaefer, and Davide Scaramuzza. 2017. Real-time Visual-Inertial Odometry for Event Cameras using Keyframe-based Nonlinear Optimization. https://doi.org/10.5244/C.31.16

[37] E. Rosten and T. Drummond. 2005. Fusing points and lines for high performance tracking. In *Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1*, Vol. 2. 1508–1515 Vol. 2. https://doi.org/10.1109/ICCV.2005.104

[38] Sourav Dey Roy and Mrinal Kanti Bhowmik. 2020. A Comprehensive Survey on Computer Vision Based Approaches for Moving Object Detection. In *2020 IEEE Region 10 Symposium (TENSYMP)*. 1531–1534. https://doi.org/10.1109/TENSYMP50017.2020.9230869

[39] Chen Song, Qixing Huang, and Chandrajit Bajaj. 2022. E-CIR: Event-Enhanced Continuous Intensity Recovery. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 7793–7802. https://doi.org/10.1109/CVPR52688.2022.00765

[40] Gary J. Sullivan, Jens-Rainer Ohm, Woo-Jin Han, and Thomas Wiegand. 2012. Overview of the High Efficiency Video Coding (HEVC) Standard. *IEEE Transactions on Circuits and Systems for Video Technology* 22, 12 (2012), 1649–1668. https://doi.org/10.1109/TCSVT.2012.2221191

[41] Bishan Wang, Jingwei He, Lei Yu, Gui-Song Xia, and Wen Yang. 2020. Event Enhanced High-Quality Image Recovery. In *European Conference on Computer Vision*. Springer.

[42] Olivia Wiles, João Carreira, Iain Barr, Andrew Zisserman, and Mateusz Malinowski. 2023. Compressed Vision for Efficient Video Understanding. In *Computer Vision – ACCV 2022*, Lei Wang, Juergen Gall, Tat-Jun Chin, Imari Sato, and Rama Chellappa (Eds.). Springer Nature Switzerland, Cham, 679–695.

[43] C. Wu, M. Zaheer, H. Hu, R. Manmatha, A. J. Smola, and P. Krahenbuhl. 2018. Compressed Video Action Recognition. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, Los Alamitos, CA, USA, 6026–6035. https://doi.org/10.1109/CVPR.2018.00631

[44] K. Xu, M. Qin, F. Sun, Y. Wang, Y. Chen, and F. Ren. 2020. Learning in the Frequency Domain. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, Los Alamitos, CA, USA, 1737–1746. https://doi.org/10.1109/CVPR42600.2020.00181

[45] Xiaokai Yi, Hanli Wang, Sam Kwong, and C.-C. Jay Kuo. 2022. Task-Driven Video Compression for Humans and Machines: Framework Design and Optimization. *IEEE Transactions on Multimedia* (2022), 1–12. https://doi.org/10.1109/TMM.2022.3233245

[46] Alex Zhu, Liangzhe Yuan, Kenneth Chaney, and Kostas Daniilidis. 2018. EV-FlowNet: Self-Supervised Optical Flow Estimation for Event-based Cameras. https://doi.org/10.15607/RSS.2018.XIV.062

[47] Sergey Zvezdakov, Denis Kondranin, and Dmitriy Vatolin. 2021. Machine-Learning-Based Method for Content-Adaptive Video Encoding. In *2021 Picture Coding Symposium (PCS)*. 1–5. https://doi.org/10.1109/PCS50896.2021.9477507

Received 29 November 2023; revised 8 February 2024; accepted 15 March 2024