Augmenting Frame-Based Vision With Temporal Context

by

Matthew Dutson

A dissertation submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy

(Computer Sciences)

at the

UNIVERSITY OF WISCONSIN–MADISON

2025

Date of final oral examination: 10/31/2025

The dissertation is approved by the following members of the Final Oral Committee:

Mohit Gupta, Professor, Computer Sciences
Yin Li, Associate Professor, Biostatistics and Medical Informatics
Xiaojin Zhu, Professor, Computer Sciences
Josiah Hanna, Assistant Professor, Computer Sciences

For Albert Schellenberg.

Acknowledgments

First, I would like to thank my advisor, Mohit Gupta, who has served as my ally and advocate throughout my graduate studies. I am immensely grateful for his kindness toward his students, who he treats as human beings first and researchers second. Likewise, thanks to Yin Li for acting as an informal co-advisor over the last few years. His expertise has been invaluable in improving the quality and impact of our work.

Thanks also to my student co-authors, including Varun Sundar, Tianyi Zhang, and Nathan Labiosa. Working with each of them and seeing their research processes up close was a tremendous learning opportunity.

I am fortunate to count my lab-mates—Aryan, Avery, Bhavya, Carter, Sacha, Shantanu, and Varun—as friends. Nobody understands the struggle like they do (and as they say, misery loves company).

Thanks to the team at Ubicept, who are as decent as they are talented. In particular, thanks to Felipe Gutierrez-Barragan for his friendship and mentorship during my time as an intern.

I am grateful to my parents, Emily and Joseph, for instilling in their children enlightened values—kindness, integrity, curiosity, generosity, resilience. It is thanks to them that I can count my siblings (Marie, John, Esther, Sam, Hannah, and Isaac) among my best friends.

I am grateful to my doctors, in particular Drs. Morgan, Golding, and Draper, for their care and support through recent health challenges. Without their help, this document would likely not exist.

Thanks to Maggie and Marzipan for making me feel important even when my papers get rejected, and for giving me an excuse to go walking.

Finally, I would like to thank my wife, Itzel. The stress of a doctorate can be contagious, and I appreciate her patience in shouldering some of that burden with me. I admire her perspective—she has a gift for pulling me down to the meaningful things happening in the real world. More than anything else, I am looking forward to spending more time with her after graduation.

This research was supported by NSF CAREER Award 1943149. The work in Chapter 2 was further supported by NSF award CNS-2107060 and Swiss National Science Foundation grant 200021_166289. The work in Chapter 5 was further supported by NFS award CPS-2333491, ARL under contract number W911NF-2020221, and the Wisconsin Alumni Research Foundation via a Research Forward Initiative.

Contents

List of Tables

List of Figures

AUGMENTING FRAME-BASED VISION WITH TEMPORAL CONTEXT

Matthew Dutson

Under the supervision of Professor Mohit Gupta
At the University of Wisconsin-Madison

A visual scene can be represented as a series of instantaneous snapshots, i.e., frames. In fact, this is often the most natural representation, as it corresponds directly to the way images are captured by a sensor. Many vision systems operate in a frame-based manner, processing each time slice independently. For example, an object detection model might be applied to each frame of a video to produce a series of predictions. There are some practical benefits to frame-based processing—for example, a single-image model requires fewer computational resources to train than a video model.

However, frame-based processing also has some drawbacks. First is bandwidth: transmitting a series of independent frames ignores temporal redundancy, limiting opportunities for compression. The second problem is compute cost: temporal redundancy implies repetitive, unnecessary computation. Third and finally, frame-by-frame processing can lead to temporal consistency, and limit a vision system's robustness against intermittent image corruptions.

In this thesis, we mitigate these drawbacks by augmenting frame-based approaches with recent temporal context. This allows us to avoid redundant bandwidth and compute, and to improve the temporal coherence of the vision system. We emphasize compatibility with existing frame-based architectures, retaining the advantages of frame-based vision (e.g., ease of training) while leveraging the information present in recent context.

Mohit Gupta

Abstract

A visual scene can be represented as a series of instantaneous snapshots, i.e., frames. In fact, this is often the most natural representation, as it corresponds directly to the way images are captured by a sensor. Many vision systems operate in a frame-based manner, processing each time slice independently. For example, an object detection model might be applied to each frame of a video to produce a series of predictions. There are some practical benefits to frame-based processing—for example, a single-image model requires fewer computational resources to train than a video model.

However, frame-based processing also has some drawbacks. First is bandwidth: transmitting a series of independent frames ignores temporal redundancy, limiting opportunities for compression. The second problem is compute cost: temporal redundancy implies repetitive, unnecessary computation. Third and finally, frame-by-frame processing can lead to temporal consistency, and limit a vision system's robustness against intermittent image corruptions.

In this thesis, we mitigate these drawbacks by augmenting frame-based approaches with recent temporal context. This allows us to avoid redundant bandwidth and compute, and to improve the temporal coherence of the vision system. We emphasize compatibility with existing frame-based architectures, retaining the advantages of frame-based vision (e.g., ease of training) while leveraging the information present in recent context.

1 Introduction

1.1 Frame-Based Vision

In the days of analog cameras, video was stored as a sequence of still frames. At the time, this was unavoidable. The camera optics directly mapped incoming rays onto the film's physical layout; "image processing" necessarily occurred in the realm of lenses and photons.

Digital photography opened a new world of possibilities, representing video as an inherently manipulable data stream. The digital revolution led to innovations including modern video compression, which achieves remarkable compression ratios in part through temporal modeling. Such methods store a video not as a sequence of independent time slices, but rather as a series of evolutions from an initial state.

Despite these changes, the frame-oriented perspective continues to inform the way we design algorithms and systems. For example, visual perception systems (e.g., for object detection or image segmentation) are commonly trained and deployed for frame-by-frame operation.

The continued prevalence of this mindset is not just a historical artifact. Compared to video, single-frame data is available at a larger scale and with greater variety, supporting the training of more accurate and robust perception models. In a similar vein, training a single-image model is less computationally inten-

sive than training a video model; the cost of time-domain training generally scales with the size of the temporal window. Finally, frame-based models offer greater flexibility than video models, supporting both single-image operation and frame-by-frame video processing.

Our goal is to *leverage temporal reasoning to improve the performance of vision systems, while retaining the advantages of frame-based processing*. We target three areas for improvement: bandwidth, compute, and stability.

Bandwidth. Of these three topics, bandwidth is arguably the most well-explored in past work. Sequential video frames are highly redundant; by detecting this redundancy and skipping repeated patterns, video codecs can compress orders of magnitude beyond single-image methods. However, cracks begin to show at high capture rates. At these speeds, unavoidable photon noise reduces the codec's ability to exploit temporal redundancy. Further, conventional codecs have a relatively high computational cost. At high speeds, the encoder may be unable to keep up with the amount of incoming data. In Chapter 2, we develop temporal compression mechanisms that are noise-tolerant, lightweight, and retain relevant high-speed information.

Compute. Temporal redundancy can also lead to unnecessary computation. In the case of (widely deployed) single-frame neural networks, this means recomputing deep features for parts of the scene that have changed very little. High-speed capture amplifies this wasted computation. At higher frame rates, the amount of visual information and change does not increase, but the computational cost scales linearly with frame rate. In Chapters 3 and 4, we propose methods to make existing neural network models "redundancy aware," so that computation scales with the amount of scene change, rather than the number of frames.

Stability and robustness. Finally, in Chapter 5, we consider how we might use temporal context to improve the *quality* of model predictions. We discuss two closely related objectives: stability and robustness. By stability, we mean

the consistency of the model's predictions over time. Robustness refers to a model's accuracy when its inputs are corrupted, for example, by adverse weather or noise. Frame-by-frame models lack temporal context; therefore, when they inevitably make errors, those errors tend to be temporally incoherent, producing outputs with jarring flickering artifacts. In this situation, considering the recent temporal context can significantly improve the stability of the outputs. Likewise, when the input is affected by a time-varying corruption, we can achieve more accurate predictions by combining information from recent frames.

Overview. In the broadest terms, our objective is this: to improve the performance of imaging systems by adding temporal context, without entirely abandoning the abstraction of frames. Especially for post-capture processing, where single-frame neural network models are prevalent, there are good reasons to start with frames—convenience, data availability, and compute costs, to name a few. Rather than replacing these models with entirely new architectures, we propose methods that *lift* frame-based vision into the video domain. Specifically, we find ways to inject lightweight logic that accounts for temporal dynamics and changes. In many cases, these modifications do not even require re-training the existing network. We hope that these approaches can serve as a general toolbox for practitioners deploying vision systems in challenging scenarios where bandwidth and compute may be limited, and where inputs may be noisy or otherwise unreliable.

In the remainder of this chapter, we continue our exploration of these three threads before diving into technical details in subsequent chapters.

1.2 Bandwidth

Conventional video pipelines, combining CMOS sensors and compression algorithms such as h264, perform well in ordinary conditions with typical

motion speeds. However, these components fail under more challenging high-speed conditions. CMOS sensors are subject to *read noise*—a noise penalty paid each time a frame is recorded. As we increase the frame rate, this read noise can overwhelm the underlying signal.

We consider an emerging type of image sensor, the SPAD (single-photon avalanche diode) [162, 194, 224], which is free of read noise and can faithfully capture scenes at very high frame rates. SPADs record the arrival of individual photons; each pixel in a SPAD frame has a binary value, with a value of 1 indicating that a photon was observed during the frame. Due to the random nature of photon arrivals, individual SPAD frames are extremely noisy. However, a *sequence* of SPAD frames provides a complete picture of a scene's evolution over time.

Due to their high frame rates, the raw output of a SPAD requires significant bandwidth and storage. Therefore, a key practical challenge in unlocking the power of these sensors is *compression*. Intuitively, we expect high-speed data to be more compressible along the time axis. Most scene content is either static or changes relatively slowly—we would like to compress (average) this content while preserving precise measurements of high-speed phenomena.

Our first thought might be to apply a mainstream video compression algorithm. However, there are two problems with this idea. First, conventional codecs are ineffective at detecting redundancy in the presence of heavy noise (as found in SPAD frames). These failures negate many of the compression gains we see on typical, less-challenging video. Second, these methods have a relatively high computational cost. Because this cost scales with the number of frames we are compressing, it would be impractical to run these codecs at the frame rate of a SPAD.

To tackle these challenges, we turn to *event-based sensing*. Instead of measuring scene intensity at regular intervals, event cameras [136, 182, 13] transmit information only when a change is detected. This behavior allows them to

preserve high time resolution without exorbitant bandwidth costs. Event detection is lightweight and can be computed near-sensor, and can be tuned for robustness against noise.

Despite commercial availability, existing event cameras have failed to gain widespread adoption. We believe this is due to a key technical limitation: current event cameras are not capable of capturing "normal" (i.e., intensity) images. Because they only transmit information when a change is detected, static and background objects are effectively invisible. Modifying event cameras to enable intensity reconstruction is non-trivial, due to the analog nature of the event-detection circuitry.

This is where single-photon sensors (SPADs) come into play. SPADs provide *digital* access to high-speed photon arrivals. This digital representation is inherently flexible—harkening back to the revolution in image representations enabled by the advent of digital photography. This flexibility allows us to define a new space of *generalized event cameras*.

In Chapter 2, we describe the space of generalized event cameras and propose several concrete designs. Our methods compress raw single-photon data by 2–3 orders of magnitude, while producing high-quality intensity images and faithfully capturing high-speed phenomena. As an additional benefit, by compressing out noise, our approach makes single-photon data more easily digestible by downstream restoration and processing models. And unlike conventional codecs, generalized events are lightweight enough to be computed at high speed near (or in) the image sensor.

1.3 Compute

The idea of event-based vision naturally extends to downstream processing models. In this case, our goal is not to reduce bandwidth but rather to reduce compute cost. There is a clear biological motivation here; mainstream deep

neural networks update neuron activations in lockstep. In the context of frame-based video processing, this means the network activations and outputs are recomputed from scratch on each frame. This approach stands in stark contrast to biological neural networks, where neurons act as independent processing nodes and communicate asynchronously via "spikes" (synaptic transmissions). This sparse, as-needed processing is one of the reasons the brain is so energy efficient.

This thinking is the driver behind an existing body of work on spiking neural networks (SNNs) [153]. Neurons in an SNN emulate biological neurons by transmitting one-bit spikes. Information is encoded in the timing and frequency of spikes.

Initially, we found ourselves drawn to this idea, especially given the conceptual similarities between one-bit neuron spikes and one-bit SPAD photon detections. In fact, we published a paper that sought to improve the practicality and real-world efficiency of SNNs [52]. However, we eventually abandoned this thread for three reasons.

First, realizing efficiency gains with an SNN requires specialized compute hardware supporting asynchronous spiking updates. This hardware is competing with mature parallel architectures (GPUs and TPUs), and it is not at all clear that it will ever outperform the alternatives in absolute terms (watts or milliseconds per frame).

Second, training SNNs presents a significant challenge. Binary spikes are not differentiable, so approximations and workarounds are required. In general, the best we can hope for is to match the performance of conventional neural networks (ANNs). Clearly, complex learning is possible in spike-driven networks—biological brains demonstrate this. However, the underlying mechanisms are likely far more sophisticated than current SNN training approaches. After all, a real biological neuron is not a mathematical abstraction (a linear function composed with a nonlinearity), but rather an intricate micro-

machine containing ∼billions of functional units (proteins).

Third and finally, it is not clear to us that there is anything special about "spiking," in the same way that "wing flapping" is not integral to flying. Many of the most accurate SNN models (including our work) encode floating-point activations as mean firing rates. Accurately representing the activation generally requires sending far more bits over the synapse than simply transmitting the underlying floating-point value.

These failings aside, the core insight underlying SNNs—that sparse, as-needed computation can improve efficiency—still holds water. Our goal is then to turn this insight into a practical method. Notably, we would like networks that are compatible with mainstream machine learning practices (reducing the risk of being left behind as the field advances). This means not using a bespoke training algorithm, and retaining compatibility with general-purpose hardware (i.e., floating-point arithmetic).

In Chapters 3 and 4, we address this challenge by presenting novel event-based processing methods for convolutional networks (Chapter 3) and vision Transformers (Chapter 4). We refer to these methods broadly as *event networks*. As the name implies, these models have many similarities with the event cameras described in Section 1.2. Event networks skip temporally redundant *computation*, analogous to how event cameras skip redundant *communication*.

An event network applies event detection across its depth, creating *event neurons* at each layer. This design is necessary because temporal redundancy can occur at various levels of visual complexity. Early layers can detect redundancy in low-level structures (e.g., pixel values or simple texture). Later layers, which operate in a higher-level space, can detect repetition in more complex patterns.

Frame-based networks update groups of neurons in lockstep. An event network is a "network" in perhaps a more literal sense—event neurons can be

viewed as independent nodes that transmit asynchronous messages. When a neuron detects that its activation has changed, it sends a message (event) to its synaptic neighbors, triggering cascading downstream updates.

We apply the event-based approach on top of an existing frame-based network. It does not require any changes to the original network's weights or representation and makes minimal assumptions about the network architecture. In this way, we ensure broad compatibility with mainstream machine learning approaches and increase the likelihood that our methods will remain relevant as the field evolves.

Our methods can yield significant computation savings (approximately an order of magnitude) while generally preserving the accuracy of the original network. Further, event networks have the intriguing properties of *adaptivity* and *controllability*. They are adaptive because the computational cost per frame varies with the scene dynamics. When there is more change (less redundancy), more computation is required, and vice versa. Event networks are controllable because we can configure the compute cost at runtime via relatively simple parameter adjustments (e.g., changing the event-firing threshold).

1.4 Stability and Robustness

Converting a model into an event network has an unexpected side effect: in many cases, it increases the temporal stability of the predictions. For example, in our experiments with pose estimation (Section 3.5.1), we observed less frame-to-frame jittering in the predicted joint positions. Event truncation can be viewed as a form of temporal smoothing, in which minor variations around a constant value are ignored.

This result points to additional drawbacks of frame-by-frame processing: temporal inconsistency. If our models were error-free, then temporal inconsistency would not be a concern—predictions would exactly match the ac-

tual world dynamics. However, current models are far from perfect. Under frame-wise processing, there is nothing keeping model errors from varying significantly between frames, with slight variations in the inputs leading to disproportionate changes in the outputs. Although the individual predictions may be mostly correct, their *dynamics* are often completely unrealistic.

Temporal incoherence is a clear problem in applications where model outputs are intended for human viewing. Human vision is highly sensitive to change, and introducing motion where it is not expected can be perceptually jarring.

For example, consider the task of denoising, where we use a noisy (e.g., low-light) image to infer a high-quality image. Denoising is an ill-posed problem, meaning the best a model can do is make an "educated guess" as to the correct image. If we naively apply a single-frame denoising model to a video, the model will not necessarily make consistent guesses across frames. The resulting output video will exhibit noticeable flickering artifacts, including in regions with little to no motion (e.g., the background). These artifacts significantly reduce the perceived quality of the video, even if the single-image predictions are generally correct.

Temporal stability is closely related to robustness under time-varying image corruptions (i.e., accurate predictions under corruptions). Consider, for example, a segmentation model operating under adverse weather conditions. Rain or snow occlude portions of the scene, making accurate predictions more challenging. However, this occlusion is *transient*—if we consider a few recent frames, we are likely to find an occlusion-free version of the same image patch. By the same token, we can improve performance under noise by considering more "samples" (frames) from recent history.

A key conceptual question here is whether we can improve stability without harming accuracy. After all, our ultimate goal is a perfectly accurate model; altering such a model to improve stability would constitute "over-smoothing reality." As part of our work in Chapter 5, we propose an objective function

that does not disturb correct ("oracle") predictions, thereby ensuring the desired behavior in the limiting case of a perfect model. We also discover a failure mode, which we refer to as "collapse," in which an overly aggressive smoothing objective causes a model to repeat its initial predictions indefinitely, regardless of its input. We use these theoretical findings to develop practical guidelines for developing stabilization methods.

Now that we have laid out our objective, we come to the question of implementation. Our goal is to take any off-the-shelf frame model and add extra logic to improve its temporal consistency and robustness. Event networks are, in fact, a good starting point—they are compatible with existing models, and already show signs of improving temporal consistency. However, here we can consider designs beyond those based on change truncation. In doing so, we can unlock new benefits such as time-differentiability, allowing us to train our stabilization mechanisms if desired.

Many stabilizer implementations would be compatible with our overall design goals. We propose a particular family of designs based on an exponential smoothing mechanism. The stabilized activation is a weighted combination of the current unstabilized activation and previous stabilized activations. This mechanism has several advantages: it does not disturb the original feature representation, has low memory overhead, and is straightforward to train.

We then introduce a controller network that dynamically predicts the degree of smoothing (i.e., the exponential weighting) for each activation value. We provide the controller with the current and previous input frames, which allow it to adjust the stabilization based on the scene content and the amount of motion. For example, the controller can apply less-aggressive smoothing along a moving edge to prevent motion blur.

By adding controlled stabilizers and training them using our proposed objective function, we achieve significant improvements in temporal stability and robustness. These improvements come without an accuracy cost—in fact, we

see substantial improvements in accuracy for many tasks. Unsurprisingly, these accuracy gains are most pronounced in tasks like denoising, where temporal context contains information directly relevant to predictions.

In summary, we show that by injecting temporal context into a frame-based model, we can make predictions better (more accurate), more stable, and more reliable. We achieve this while retaining broad compatibility with existing single-image models. In this way, we keep the advantages of the frame-based approach, including more readily available training data, a wider selection of models, and less resource-intensive training and inference.

1.5 Summary

Frame-centric approaches are still prevalent in deployed vision systems. However, emerging technologies, including single-photon cameras and large neural networks, push frame-based methods to their limits—inspiring us to develop new strategies.

On the sensing side (Chapter 2), we consider emerging single-photon sensors that produce highly noisy data at extreme frame rates. Representing this data in its raw form is impractical, as it quickly exceeds downstream memory and storage limits. Instead, we propose a family of event-based codecs that leverage *temporal context* to reduce bandwidth. Unlike conventional codes, our methods are lightweight and noise-tolerant, providing significant compression while producing high-quality images.

On the processing side, we leverage temporal context to achieve both computation savings (Chapter 3 and 4) and improved stability and robustness (Chapter 5). A recurring theme in these works is the injection of *recurrences* into existing single-frame models to impart a desired property. By injecting these recurrences at all network depths, we enable reasoning in both the low-level input space and the abstract feature space. Because we are not proposing

a specific architecture but rather a family of techniques, our methods have the potential for broad impact and continued relevance as future architectural advancements unfold.

In the following chapters, we provide the complete details of our methods and experiments demonstrating their effectiveness. In the last chapter (Chapter 6), we offer some final discussion and directions for future work.

2 Bandwidth: Generalized Event Cameras

In Chapters 3 and 4, we leveraged repetition between frames to reduce computational costs. In this chapter, we explore another potential benefit of redundancy-aware vision: data compression. This idea has been explored extensively in the context of mainstream video codecs (e.g., h264), which achieve impressive compression ratios on video from conventional cameras. However, data from emerging single-photon sensors presents unique challenges: it is computationally infeasible to run conventional codecs at high single-photon frame rates, and the extreme noise in individual frames makes codec heuristics less effective.

We explore an alternative compression approach based on *event-based sensing*. Existing event cameras capture the world at high time resolution and with minimal bandwidth requirements. However, their event streams, which only encode changes in brightness, do not contain sufficient scene information to support a wide variety of downstream tasks. In this chapter, we design *generalized* event cameras that preserve scene intensity in a bandwidth-efficient manner. We generalize event cameras in terms of when an event is generated and what information is transmitted. Single-photon sensors, by providing digital access to individual photon detections, give us the flexibility to realize a rich space of generalized event cameras.

Our single-photon event cameras are capable of high-speed, high-fidelity imaging at low readout rates. Consequently, they can support plug-and-play downstream inference, without capturing new event datasets or designing specialized event-vision models. As a practical implication, our designs, which involve lightweight and near-sensor-compatible computations, provide a way to use single-photon sensors without exorbitant bandwidth costs.

2.1 Introduction

Event cameras [136, 182, 13] sense the world at high speeds, providing visual information with *minimal bandwidth and power*. They achieve this by transmitting only changes in scene brightness, when significant "events" occur. However, there is a cost. Raw event data, a sparse stream of binary values, does not hold sufficient information to be used directly with mainstream vision algorithms. Therefore, while event cameras have been successful at certain tasks (e.g., object tracking [62, 128], obstacle avoidance [78, 16, 200], and high-speed odometry [31, 277, 82]), they are not widely deployed as general-purpose vision sensors, and often need to be supplemented with conventional cameras [62, 82, 61]. These limitations are holding back this otherwise powerful technology.

Is it possible to realize the promise of event cameras, i.e., high temporal resolution at low bandwidth, *while preserving* rich scene intensity information? To realize these seemingly conflicting goals, we propose a novel family of *generalized event cameras*. We conceptualize a space of event cameras along two key axes (Figure 2.1 (top)): (a) "when to transmit information," formalized as a change detection procedure Δ; and (b) "what information to transmit," characterized by an integrator Σ that encodes incident flux. Existing event cameras represent one operating point in this (Σ, Δ) space. Our key observation is that by exploring this space and considering new (Σ, Δ) combinations, we can design event cameras that preserve scene intensity. We propose more

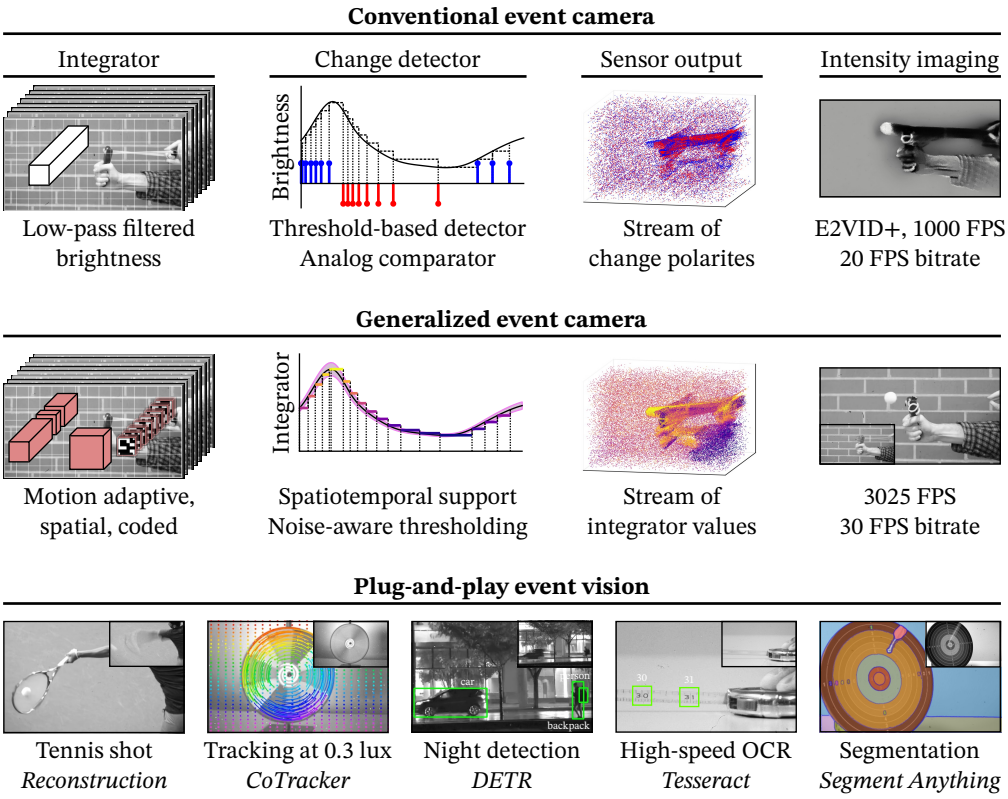


Figure 2.1: **Generalized event cameras. (top)** Event cameras generate outputs in response to abrupt changes in scene intensity. We describe this as a combination of a low-pass integrator and a threshold-based change detector. **(middle)** We generalize the space of event cameras by designing *integrators* that capture rich intensity information, and more reliable *change detectors* that utilize larger spatiotemporal contexts and noise-aware thresholding (Sections 2.4.1, 2.4.2, and 2.4.3). Unlike existing events, our generalized event streams inherently preserve scene intensity, e.g., this ping-pong ball slingshotted against a brick wall backdrop. **(bottom)** Generalized event cameras enable high-fidelity bandwidth-efficient imaging: providing 3025 FPS reconstructions with a readout equivalent to a 30 FPS camera. Consequently, generalized events facilitate *plug-and-play inference* on a multitude of tasks in challenging scenarios (insets depict the extent of motion over 30 ms).

general integrators, e.g., that represent flux according to motion levels, that span spatial patches, or that employ temporal coding (Figure 2.1 (middle)). We also introduce robust change detectors that better distinguish motion from noise, by considering increased spatiotemporal contexts and modeling noise in the sensor measurements.

Despite their conceptual appeal, physically implementing generalized event cameras is a challenge. This is because the requisite computations must be performed *at the sensor* to achieve the desired bandwidth reductions. For example, existing event cameras perform simple integration and thresholding operations via analog in-pixel circuitry. However, more general (Σ, Δ) combinations are not always amenable to analog implementations; even feasible designs might require years of hardware iteration and production scaling. To physically realize generalized event cameras, we leverage an emerging sensor technology: single-photon avalanche diodes (SPADs) that provide *digital access* to photon detections at extremely high frame rates. This allows us to compose arbitrary *software-level* transformations, such as those required by generalized event cameras. Further, we are not locked to a particular event camera design and can realize multiple configurations with the same sensor.

Implications: extreme, bandwidth-efficient vision. Generalized event cameras support high-speed, high-quality image reconstruction, but at low bandwidths quintessential of current event cameras. For example, in Figure 2.1 (middle, bottom), we show reconstructions at 3025 FPS that have an effective readout of a 30 FPS frame-based camera. Further, our methods have strong low-light performance due to the SPAD's single-photon sensitivity. As we show in Figure 2.1 (bottom), preserving scene intensity facilitates plug-and-play inference in challenging scenarios, with state-of-the-art vision algorithms. Critically, this does not require retraining vision models or curating dedicated datasets, which is a significant challenge for unconventional imagers. This plug-and-play capability is vital to realizing universal event vision that retains the benefits of current event cameras.

Scope. We consider *full-stack* event perception: we conceptualize a novel space of event cameras, provide relevant single-photon algorithms, analyze their imaging capabilities and rate-distortion trade-offs, and show on-chip feasibility. We demonstrate imaging capabilities in Sections 2.5.1 and 2.5.2 using the SwissSPAD2 array [224], and show viable implementations of our algorithms for UltraPhase [7], a recent single-photon compute platform. All of these are critical to unlocking the promise of event cameras. However, our objective is not to develop an integrated system that incorporates all these components; this work merely takes the first steps toward that goal.

2.2 Related Work

Event camera designs. Perhaps most widespread is the DVS event camera [136], where each pixel generates an event in response to measured changes in (log) intensity. The DAVIS event camera [20, 13] couples DVS pixels with conventional CMOS pixels, providing access to image frames. However, the frames lack the dynamic range of DVS events. A recent design, Celex-V [90], provides log-intensity frames using an external trigger. ATIS [182] features asynchronous intensity events, but its sophisticated circuitry reduces pixel fill factor. The above designs are based on analog processing; we instead design event cameras on *digital* photon detections.

Intensity imaging with event cameras. Several approaches have been explored to obtain images from events, including Poisson solvers [11], manifold regularization [164], assuming knowledge of camera motion [112, 42] or optical flow [271], and learning-based methods [189, 202, 278, 214, 175]. However, because events often lack sufficient scene information, they are often supplemented by conventional frames [201, 21, 206, 172], either from sensors such as DAVIS or using a multi-camera setup. Fusing events and frames presents challenges due to potential spatiotemporal misalignment and discrepancies in imaging modalities. Even when these challenges are over-

come, we show that fusion methods produce lower fidelity than our proposed generalized event cameras.

Passive single-photon imaging. In the past few years, SPADs have found compelling passive imaging applications; this includes high-dynamic range imaging [96, 95, 145, 165], motion deblurring [97, 151, 152, 120, 121], high-speed tracking [72], and ultra wide-band videography [242]. A particularly relevant method is proposed by Seets et al. [204], which uses flux changepoint estimation to perform burst photography on single-photon sequences. This approach uses flux changepoints to estimate motion, then integrates along spatiotemporal motion trajectories to circumvent the noise-blur tradeoff. This spatiotemporal integration allows for high-quality reconstructions under challenging lighting and motion conditions. In contrast, our work emphasizes changepoint estimation as a means to compress single-photon data. Further, since we aim to run our proposed techniques near sensor, where there are limited memory and compute capabilities, we focus on online changepoint estimation that processes photon detections in a single pass.

The fine granularity of passive single-photon acquisition makes it possible to emulate a diverse set of imaging modalities [217], including event cameras, via post-capture processing. In this work, we go beyond emulating existing event cameras and design alternate event cameras that preserve high-fidelity intensity information.

2.3 What is an Event Camera?

The defining characteristic of event cameras is that they transmit information *selectively*, in response to changes in scene content. This selectivity allows event cameras to encode scene information at high time resolutions required to capture scene dynamics, *without* proportionately high bandwidth. This is in contrast to frame-based cameras, where readout occurs at fixed intervals.

We characterize event cameras in terms of two axes: *what* the camera transmits and *when* it transmits. As a concrete example, consider existing event cameras. They trigger events ("when to transmit") based on a fixed threshold:

$$|\Phi(\mathbf{x}, t) - \Phi_{\text{ref}}(\mathbf{x})| \geq \tau, \tag{2.1}$$

where $\Phi(\mathbf{x}, t)$ is a flux estimate at pixel $\mathbf{x}$ and time t, and τ is the threshold. Event cameras such as the DVS [20] measure a temporally low-pass filtered estimate of log-flux; we absorb this into $\Phi(\mathbf{x}, t)$ for brevity. $\Phi_{\text{ref}}(\mathbf{x})$ is a previously-recorded reference, set to $\Phi(\mathbf{x}, t)$ whenever an event is triggered. Each event consists of a packet

$$(\mathbf{x}, t, \text{sign}(\Phi(\mathbf{x}, t) - \Phi_{\text{ref}}(\mathbf{x}))) \tag{2.2}$$

that encodes the polarity of the change ("what to transmit").

Event polarities, although adequate for some applications, do not retain sufficient information to support a general set of computer vision tasks. A stream of event polarities is an extremely lossy representation. Notably, it only defines scene intensity up to an initial unknown reference value, and it does not encode any information in regions not producing events, i.e., regions with little or no motion.

Our key observation is that existing event cameras represent just one operating point in a broader space of *generalized event cameras*, which is defined by two axes: "what to transmit" and "when to transmit." By considering alternate points in this space, we can design event cameras that preserve high-fidelity scene intensity. This enables plug-and-play inference with a host of algorithms developed by the mainstream vision community.

We begin with a conceptual exploration of this generalized space, before describing its physical implementation.

1-bit events | Flux levels | Adaptive exposures

Figure 2.2: **Altering "what to transmit." (left)** We sum the events generated by a jack-in-the-box toy as it springs up. This sum gives a lossy encoding of brightness changes in dynamic regions. **(middle)** Transmitting levels instead of changes helps recover details in static regions. **(right)** Adaptive exposures, which accumulate flux between events, provide substantial noise reduction.

Generalizing "what to transmit." As a first step, we can modify the event camera such that it transmits n-bit values instead of one-bit change polarities. When an event is triggered, we send the current value of $\Phi(\mathbf{x}, t)$; if a pixel triggers no events, we transmit Φ during the final readout. As we show in Figure 2.2 (b), this simple change allows us to recover scene intensity, even in static regions. It is important to note that, while the transmitted quantity differs from conventional events, we retain the defining feature of an event camera: selective transmission based on scene dynamics (where we transmit according to Equation 2.1). Thus, the readout remains decoupled from the time resolution.

If $\Phi(\mathbf{x}, t)$ were a perfect estimate of scene intensity, then the changes thus far would suffice. However, Φ is fundamentally noisy: to capture high-speed changes, Φ must encompass a shorter duration, which leads to higher noise. This is a manifestation of the classical noise-blur tradeoff.

To address this problem, we introduce the abstraction of an *integrator* (or Σ), that defines how we accumulate incident flux and, in turn, what we transmit. Ideally, we want the integrator to *adapt* to scene dynamics, i.e., accumulate over longer durations when there is less motion, and vice versa. We observe that event generation, which is based on scene dynamics, can be used to

formulate an adaptive integrator. Specifically, we propose an integrator Σ_{cuml} that computes the cumulative flux since the last event:

$$\Sigma_{\text{cuml}}(\mathbf{x}, t) = \int_{T_0}^{t} \Phi(\mathbf{x}, s) ds, \tag{2.3}$$

where T_0 is the time of the last event. When an event is triggered at time T_1, we communicate the value of $\Sigma_{\text{cuml}}(\mathbf{x}, T_1)$, which we interpret as the intensity throughout $[T_0, T_1]$ (we can either transmit values of Σ_{cuml} or changes to Σ_{cuml}; we treat this as an implementation detail here). This approach yields a piecewise constant time series, with segments delimited by events. We note that a similar idea, of virtual exposures beginning and ending with change points, was also explored in [204] as part of a motion-adaptive deblurring pipeline. Adaptive exposures significantly reduce noise while preserving dynamic scene content, as we show in Figure 2.2 (right).

Generalizing "when to transmit." The success of the adaptive integrator crucially depends on the reliability of events; for example, triggering false events in static regions causes unnecessary noise. We refer to the event-generation procedure as the *change detector*, denoted by Δ. Current event cameras detect changes by applying a fixed threshold to measured intensity differences (Equation 2.1). This method has two key limitations: it only considers the value of Φ at pixel location $\mathbf{x}$ and time t and is not attuned to the stochasticity in Φ.

We design more robust change detectors that (1) leverage enhanced spatiotemporal contexts, and (2) incorporate noise awareness, either explicitly by tuning thresholds, or implicitly by modulating the detector's behavior. Specifically, we improve reliability by using temporal forecasters (Section 2.4.1), by leveraging correlated changes in patches (Section 2.4.2), or by exploiting integrator statistics (Section 2.4.3).

Realizing generalized event cameras. The critical detail remaining is how

we implement our proposed designs in practice. We need direct access to flux estimates at a high time resolution. Conventional high-speed cameras can provide such access, however, they incur substantial per-frame read noise ($\sim$20–40e^- [94]) that grows with frame rate [19].

We turn to an emerging class of sensors, single-photon avalanche diodes (SPADs [194]), that has witnessed dramatic improvements in device practicality and key sensor characteristics (e.g., array sizes and fill factors) in recent years [224, 162]. SPADs can operate at extremely high speeds ($\sim$100 kHz) without incurring per-frame read noise. Each $\Phi(\mathbf{x}, t)$ measured by a SPAD is limited only by the fundamental stochasticity of photon arrivals (shot noise). This allows SPADs to provide high timing resolution without a substantial noise penalty. In the next section, we describe the image formation model of SPADs and provide single-photon implementations of our designs.

2.4 Single-Photon Generalized Event Cameras

A SPAD array can operate as a high-speed photon detector, producing binary frames as output. Each binary value indicates whether at least one photon was detected during an exposure. The SPAD output response, $\Phi(\mathbf{x}, t)$, can be modeled as a Bernoulli random variable, with

$$P\left(\Phi(\mathbf{x}, t) = 1\right) = 1 - e^{-N(\mathbf{x},t)}, \tag{2.4}$$

where $N(\mathbf{x}, t)$ is the average number of photo-electrons during an exposure, including any spurious detections. The *inherently digital* SPAD response allows us to compute software-level transformations on the signal $\Phi(\mathbf{x}, t)$, including operations that may be challenging to realize via analog processing. These transformations can be readily reconfigured, which permits a spectrum of event camera designs, not just one particular choice. However, there is one consideration: our designs should be lightweight and computable on chip.

As we show in Section 2.5.4, this is vital to implementing generalized event cameras without the costs associated with reading off raw SPAD outputs.

We now describe a set of SPAD-based event cameras, beginning with the adaptive exposure method from the previous section.

Adaptive-exposure event camera. We obtain a SPAD implementation of the adaptive exposure described in Equation 2.3 by replacing the integral with a sum over photons:

$$\Sigma_{\text{cuml}}(\mathbf{x}, t) = \sum_{s=T_0}^{t} \Phi(\mathbf{x}, s). \tag{2.5}$$

To generate events, we can use a threshold-based change detector (Equation 2.1). Differences between individual binary values are not sufficiently informative; therefore, we apply Equation 2.1 to an exponential moving average (EMA) computed on Φ. We call this an "adaptive-EMA" event camera.

2.4.1 Bayesian Change Detector

A fixed-threshold change detector such as Equation 2.1 does not account for the SPAD's image formation model; it uses the same threshold irrespective of the underlying variance in photon detections. As a result, such a detector may fail to detect changes in low-contrast regions without producing a large number of false-positive detections (see Figure 2.3 (left)).

In this section, we consider a Bayesian change detector, BOCPD [1], that is tailored to the Bernoulli statistics of photon detections. BOCPD uses a series of *forecasters* to estimate the likelihood of an abrupt change. At each time step, a new forecaster ν_t is initialized as a recurrence of previous forecasters,

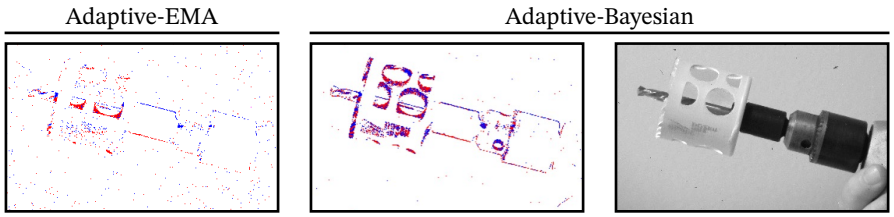


Figure 2.3: **Bayesian change detector**. **(left)** A fixed-threshold change detector (used in adaptive-EMA) makes it difficult to segment low-contrast changes. **(center)** The Bayesian detector attunes to the stochasticity in incident flux and can detect fine-grained changes such as the corners of the hole saw bit; **(right)** as a result, the integrator captures the rotational dynamics.

and existing forecasters are updated:

$$\begin{aligned} \nu_t &= (1-\gamma)\sum_{s=1}^{t-1} l_s \nu_s \\ \nu_s &\leftarrow \gamma l_s \nu_s \;\; \forall\, s < t, \end{aligned} \tag{2.6}$$

where $\gamma \in [0,1]$ is the sensitivity of the change detector, with larger γ resulting in more frequent detections. l_s is the predictive likelihood of each forecaster, which we compute by tracking two values per forecaster, α_s and β_s, that correspond to the parameters of a Beta prior. For a new forecaster, these values are initialized to 1 each, reflecting a uniform prior. Existing (α_s, β_s), $\forall\, s < t$, are updated as

$$\begin{aligned} \alpha_s &\leftarrow \alpha_s + \Phi(\mathbf{x}, t) \\ \beta_s &\leftarrow \beta_s + 1 - \Phi(\mathbf{x}, t). \end{aligned} \tag{2.7}$$

l_s is given by $\alpha_s/(\alpha_s+\beta_s)$ if $\Phi(\mathbf{x},t) = 1$, and $\beta_s/(\alpha_s+\beta_s)$ otherwise. An event is triggered if the highest-value forecaster does not correspond to T_0, the timestamp of the last event; mathematically, if $\text{argmax}_t\, \nu_t \neq T_0$.

To make BOCPD viable in memory-constrained scenarios, we apply extreme pruning by retaining only the three highest-value forecasters [240]. We also in-

corporate restarts, deleting previous forecasters when a change is detected [4].

Compared to an EMA-based change detector, the Bayesian approach more reliably triggers events in response to scene changes while better filtering out stochastic variations caused by photon noise—which we show in Figure 2.3.

2.4.2 Spatiotemporal Chunk Events

Section 2.4.1 leverages an expanded *temporal* context for change detection; however, it treats each pixel independently and does not exploit *spatial* information. In this section, we devise an event camera with enhanced spatial context that operates on small patches, e.g., of 4×4 pixels. It is difficult to derive efficient Bayesian change detectors for multivariate time series; thus, we adopt a *model-free* approach that does not explicitly parameterize the patch distribution. To afford computational breathing room for more expensive patch-wise operations, we employ temporal chunking. That is, we average $\Phi(\mathbf{x}, t)$ over a small number of binary frames (e.g., 32 binary frames) instead of operating on individual binary frames; generally, this averaging does not induce perceptible blur.

Let vector $\mathbf{\Phi}_{\text{chunk}}(\mathbf{y}, t)$ represent the chunk-wise average of photon detections at patch location $\mathbf{y}$. Let vector $\mathbf{\Sigma}_{\text{patch}}(\mathbf{y}, t)$ be an integrator representing the cumulative mean since the last event, but excluding $\mathbf{\Phi}_{\text{chunk}}$. We want to estimate whether $\mathbf{\Phi}_{\text{chunk}}$ belongs to the same distribution as $\mathbf{\Sigma}_{\text{patch}}$. We do so with a lightweight approach, that computes the distance between $\mathbf{\Phi}_{\text{chunk}}$ and $\mathbf{\Sigma}_{\text{patch}}$ in the linear feature space of matrix $\mathbf{P}$. As we show in Figure 2.4, linear features allow us to capture spatial structure within a patch. Geometrically, $\mathbf{P}$ induces a hyperellipsoidal decision boundary, in contrast to the spherical boundary of the L2 norm.

This method generates an event whenever

$$||\mathbf{P}(\tilde{\mathbf{\Phi}}_{\text{chunk}}(\mathbf{y}, t) - \tilde{\mathbf{\Sigma}}_{\text{patch}}(\mathbf{y}, t))||_2 \geq \tau, \tag{2.8}$$

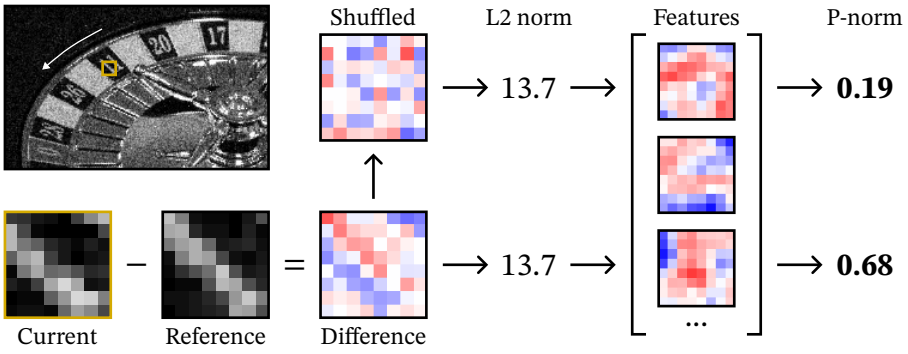


Figure 2.4: **Spatiotemporal chunk events.** We evaluate the difference between the current chunk and a stored reference in a learned linear-feature space. Unlike the L2 norm, which is permutation-invariant, the feature-space norm is sensitive to spatial structure. Randomly shuffling the pixel values reduces the transform-domain norm (the shuffled patch has a more "noise-like" structure).

where τ is the threshold. When there is no event, we extend the cumulative mean to include the current chunk. Before computing linear features, we normalize $\mathbf{\Phi}_{\text{chunk}}$ and $\mathbf{\Sigma}_{\text{patch}}$ element-wise according to the estimated variance in $\mathbf{\Phi}_{\text{chunk}} - \mathbf{\Sigma}_{\text{patch}}$; we annotate the normalized versions with a tilde. We estimate the variance based on the fact that, in a static patch, the elements of $\mathbf{\Phi}_{\text{chunk}}$ and $\mathbf{\Sigma}_{\text{patch}}$ are independent binomial random variables.

We train the matrix $\mathbf{P}$ on simulated SPAD data, generated from interpolated high-speed video. We apply backpropagation through time to minimize the MSE error of the transmitted patch values. To address the non-differentiability arising from the threshold, we employ surrogate gradients. Please see Section A.4 for complete details of this method.

2.4.3 Coded-Exposure Events

In this section, we design a generalized event camera by applying change detection to coded exposures [186, 83], which capture temporal variations by

multiplexing photon detections over an integration window. This is interesting in two aspects. First, we are designing event streams based on an modality not typically associated with event cameras. Second, we show that high-speed information can be obtained even when the change detector operates at a coarser time granularity. Coded-exposure events provide somewhat lower fidelity than our designs in Sections 2.4.1 and 2.4.2, but are more compute- and power-efficient, owing to less frequent execution of the change detector.

At each pixel, we multiplex a temporal chunk of T_{code} (∼256–512) binary values with a set of J (∼2–6) codes $C^j(\mathbf{x}, t)\,\forall\, 1 \leq j \leq J$, producing J coded exposures

$$\Sigma^j_{\text{coded}}(\mathbf{x}, t) = \sum_{s=t-T_{\text{code}}}^{t} \Phi(\mathbf{x}, s) C^j(\mathbf{x}, s). \tag{2.9}$$

The codes C^j are chosen to be random, mutually orthogonal binary masks, each containing $T_{\text{code}}/\max(2, J)$ ones [217].

We exploit the statistics of coded exposures to derive a change detector. Observe that in static regions, $\Sigma^j_{\text{coded}}(\mathbf{x}, t)$ are independent and identically distributed (iid) binomial random variables. Thus, we can expect them to lie within a binomial confidence interval of one another. If not, we assume the pixel is dynamic and generate an event. We trigger an event if $\Sigma^j_{\text{coded}} \notin \text{conf}(n, \hat{p})$ for any j. Here, "conf" refers to a binomial confidence interval (e.g., Wilson's score), $n = T_{\text{code}}/J$ draws, and $\hat{p} = \sum_s \Phi(\mathbf{x}, s)/T_{\text{code}}$ is the empirical success probability.

If a pixel is static, we store the sum of the J coded exposures, which is a long exposure, denoted by Σ_{long}. If the pixel remains static across more than one temporal chunk, we extend Σ_{long} to include the entire duration. Whereas, if the pixel is dynamic, we transmit $\left\{\Sigma^j_{\text{coded}}\right\}$, as well as any previous static intensity encoded in Σ_{long}. Downstream, we can apply coded-exposure restoration techniques [256, 207] to recover intensity frames from coded measurements.

2.5 Experimental Results

We demonstrate the capabilities of generalized event cameras using a SwissSPAD2 array [224] with resolution 512×256, which we use to capture one-bit frames at 96.8 kHz. We show the feasibility of our designs on UltraPhase [7], a recent single-photon computational platform (Section 2.5.4).

Refinement model. For each of our event cameras, we train a refinement model that mitigates artifacts arising from the asynchronous nature of events. This model takes a periodic frame-based sampling of integrator values and outputs a video reconstruction. The sampling rate is configurable; in practice, we set it ∼16–64× lower than the SPAD rate. We use a densely-connected residual architecture [232], trained on data generated by simulating photon detections on temporally interpolated [91] high-speed videos from the XVFI dataset [212]. See Section A.4 for training details.

2.5.1 Extreme Bandwidth-Efficient Videography

High-speed videography. In Figure 2.5, we capture the dynamics of a deformable ball (a "stress ball") using a SPAD, a high-speed camera (Photron Infinicam) operated at 500 FPS, and a commercial event camera (Prophesee EVK4). The high-speed camera suffers from low SNR due to read noise, which manifests as prominent artifacts after on-camera compression. Meanwhile, conventional events captured by Prophesee, when processed by "intensity-from-events" methods such as E2VID+ [213] fail to recover intensities reliably, especially in static regions. We also evaluate EDI [171], a hybrid event-frame method. We consider an idealized variant that operates on SPAD events (obtained via EMA thresholding), which gives perfect event-frame alignment and a precisely known event-generation model. We refine the outputs of EDI using the same model as for our methods. We refer to this idealized, refined version of EDI as "EDI++." While EDI++ recovers more detail than other

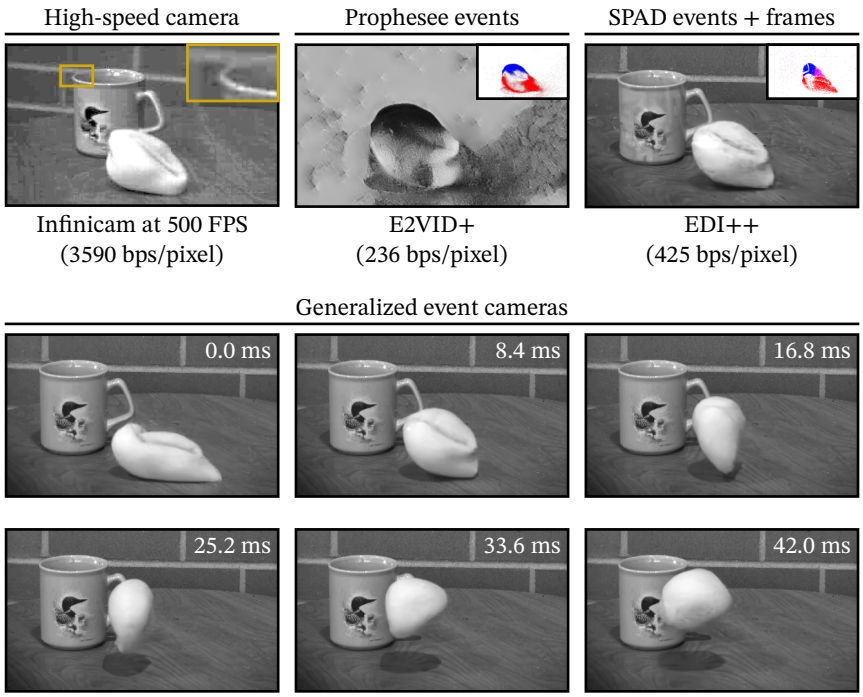


Figure 2.5: **High-speed videography. (top row)** This indoor scene is challenging for existing imaging systems, including: high-speed cameras (SNR-related artifacts), event cameras (poor restoration quality), and even hybrid event + frame techniques (reconstruction artifacts). **(bottom rows)** In contrast, our generalized event cameras capture the stress ball's extensive deformations with high fidelity and an efficient readout.

baselines, there are considerable artifacts in its outputs.

Our method achieves high-quality reconstructions at 3025 FPS (96800/32) that faithfully capture non-rigid deformations, with only 431 bits per second per pixel (bps/pixel) readout, which is a 227× compression (96800/431) of the raw SPAD capture. Viewed differently, for a 1 MPixel array, we would obtain a bitrate of 431 Mbps, implying that we can read off these 3025 FPS reconstructions over USB 2.0 (which supports transfer speeds of up to 480 Mbps).

Event imaging in low light. Figure 2.6 compares the low-light performance of frame-based, event-based, and a generalized event camera on an urban night-time scene at 7 lux (lux measured at the sensor). For frame-based cameras, a short exposure that preserves motion may be too noisy, while a long exposure can be severely blurred. The Prophesee's performance deteriorates in low light, resulting in blurred temporal gradients. EDI++, benefiting from the idealized SPAD-based implementation, can image this scene, but finer details like the motorcyclist are lost. Our generalized event cameras, on the other hand, provide reconstructions with minimal noise, blur, or artifacts—while retaining the bandwidth efficiency of event-based systems. The compression here is 307× with respect to raw SPAD outputs.

2.5.2 Plug-and-Play Inference

Generalized event cameras preserve scene intensity, which enables plug-and-play event-based vision. We consider a tennis sequence (of 8196 binary frames) containing a range of object speeds. We evaluate a range of tasks: pose estimation (HRNet [216]), corner detection [75], optical flow (RAFT [221]), object detection (DETR [29]), and segmentation (SAM [115]). We compare against event-based methods on Prophesee events; we use Arc* [5] for corner detection and E-RAFT [63] for flow. For other tasks, which do not have equivalent event methods, we run HRNet, DETR, and SAM on E2VID+ outputs.

As Figure 2.7 (middle) shows, traditional events are bandwidth efficient (331 bps/pixel), but do not provide sufficient information for successful inference. Generalized events (bottom) have a somewhat higher readout of 520 bps/pixel, but support accurate inference without requiring dedicated algorithms. To provide context for these rates, we compare them against frame-based methods (top). A long exposure (120 bps/pixel) blurs out the racket. Burst methods [76] recover a sharp image from a stack of short exposures, but with a large readout of 15100 bps/pixel.

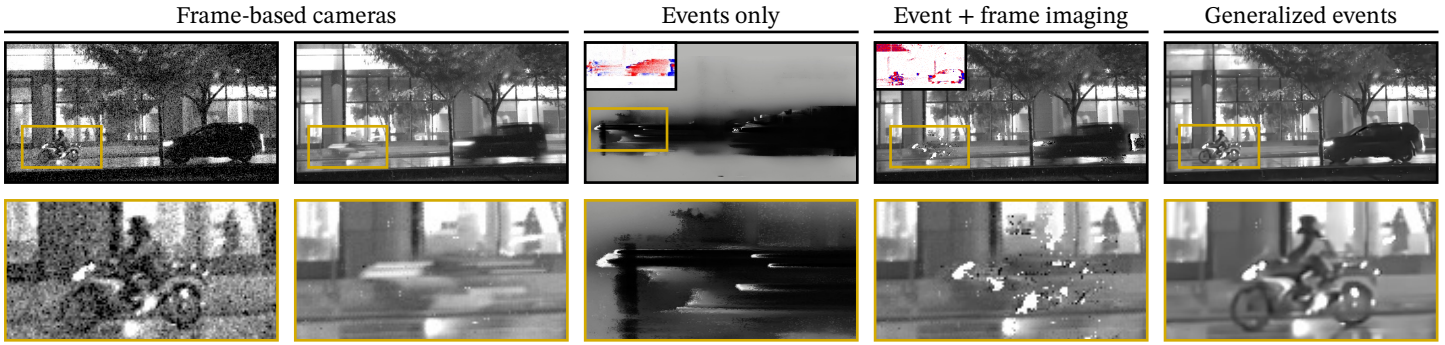


Figure 2.6: **Event imaging at night. (left to right)** Low-light conditions (7 lux, sensor side) necessitate long exposures in frame-based cameras, resulting in unwanted motion blur. The Prophesee EVK4 suffers from severe degradation in low light, causing E2VID+ to fail. Running EDI++ on perfectly aligned SPAD-frames and -events improves overall restoration quality but still gives failures on fast-moving objects. Our generalized events recover significantly more detail in low light, as seen in the inset of the motorcyclist.

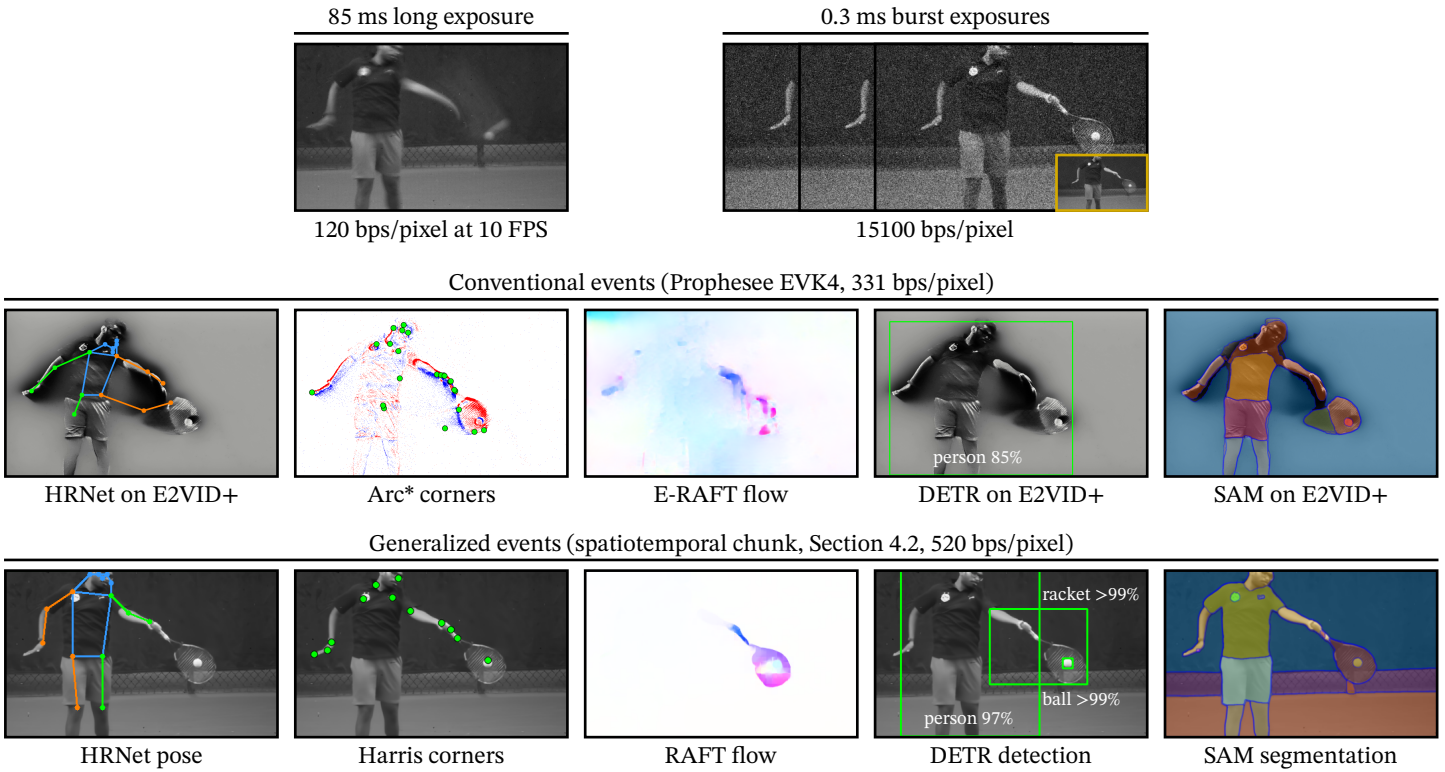


Figure 2.7: **Plug-and-play inference. (middle)** Conventional events encode temporal-gradient polarities; this lossy representation limits performance on downstream tasks. **(bottom)** Generalized events encode rich scene-intensity information, with a readout comparable existing event cameras. They facilitate high-quality plug-and-play inference, without requiring dedicated algorithms. **(top)** Generalized event cameras give image quality comparable to burst photography techniques that require much more bandwidth.

2.5.3 Rate-Distortion Analysis

Each method in Section 2.4 features a sensitivity parameter that controls the sensor readout rate (event rate), which in turn influences image quality.In this subsection, we evaluate the impact of readout on image quality (PSNR) by performing a rate-distortion analysis. For ground truth, we use a set of YouTube-sourced high-speed videos captured by a Phantom Flex4k at 1000 FPS; see the supplement to our paper [218] for thumbnails and links. We upsample these videos to the SPAD's frame rate and then simulate 4096 binary frames using the image formation model described in Equation 2.4. When computing readout for our methods, we assume that events encode 10-bit values and account for the header bits of each event packet.

As baselines, we consider EDI++, a long exposure, compressive sensing with 8-bucket masks, and burst denoising [76] using 32 short exposures. As Figure 2.8 shows, generalized event cameras provide a 4–8 dB PSNR improvement over baseline methods. Further, our methods can compress the raw SPAD response by around 80× before a noticeable drop-off in PSNR is observed.

Among our methods, the spatiotemporal chunk approach of Section 2.4.2 gives the best PSNR, followed by the Bayesian method (Section 2.4.1) and coded-exposure events (Section 2.4.3). That said, all methods are fairly similar in terms of rate-distortion (e.g., all three give comparable results for the scenes in Sections 2.5.1 and 2.5.2). The methods are better distinguished by their practical characteristics. The Bayesian method gives single-photon temporal resolution; however, as we show in Section 2.5.4, it is the most expensive to compute on-chip. The chunk-based method occupies a middle ground in terms of latency and cost. Coded-exposure events have the highest latency—events are generated only every ∼256–512 binary frames—but the lowest on-chip cost. This provides an end user the flexibility to choose from the space of generalized event cameras based on the latency requirements and the compute constraints of the target application.

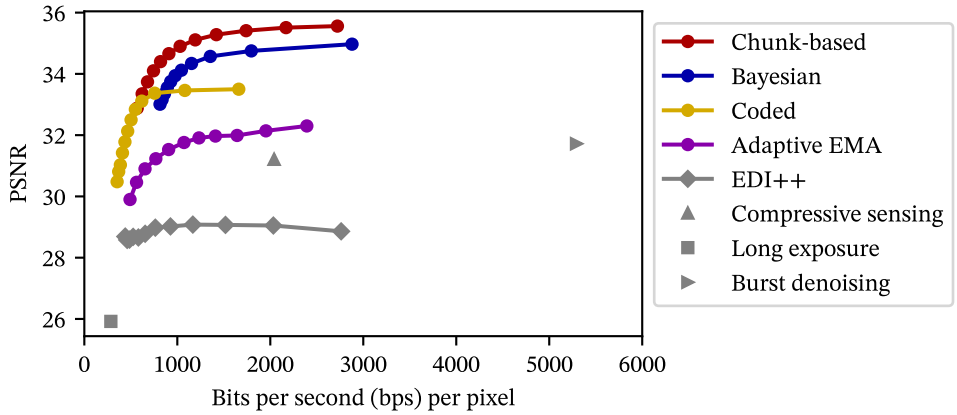


Figure 2.8: **Rate-distortion evaluation.** Our techniques feature a tunable parameter that controls the output event rate. Generalized events offer a 4–8 dB improvement in PSNR over EDI++ (at the same readout), and can compress raw photon data by 80×.

2.5.4 On-Chip Feasibility and Validation

A critical limitation of single-photon sensors is the exorbitant bandwidth and power costs involved in reading off raw photon detections. However, the lightweight nature of our event camera designs allows us to sidestep this limitation by performing computations on-chip. We demonstrate that our methods are feasible on UltraPhase [7], a SPAD compute platform. UltraPhase consists of 3×6 compute cores, each of which is associated with 4×4 pixels.

We implement our methods for UltraPhase using custom assembly code. Some methods require minor modifications due to instruction-set limitations; see Section A.9 for details. We process 2500 SPAD frames from the tennis sequence used in Section 2.5.2, cropped to the UltraPhase array size of 12×24 pixels. We determine the number of cycles required to execute the assembly code and estimate the chip's power consumption and readout bandwidth.

The coded-exposure method is particularly efficient; on most binary frames, it only requires multiplying a binary code with incident photon detections.

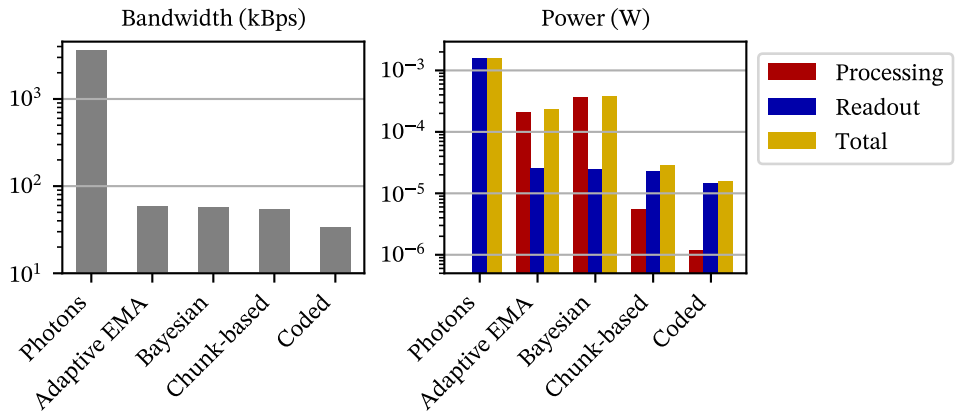


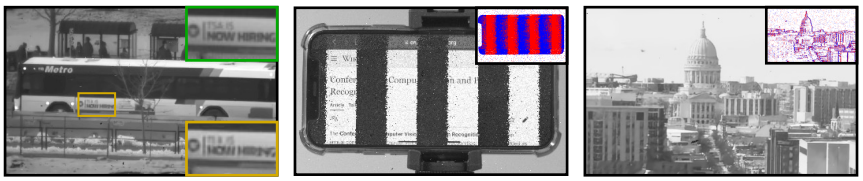
Figure 2.9: **On-chip compatibility.** We validate the feasibility of our approach on UltraPhase [7], a computational SPAD imager. Compared to reading out raw photon data, all of our approaches give marked reductions in both bandwidth **(left)** and power **(right)**.

Our proof-of-concept evaluation may pave the way for future near-sensor implementations of generalized event cameras, which with advances in chip-to-chip communication, could involve a dedicated "photon processing unit," similar to a camera image signal processor (ISP).

2.6 Limitations and Discussion

Generalized events push the frontiers of event-based imaging; however, some scenarios lead to sub-optimal performance. As seen in Figure 2.10 (left), if the scene dynamics is entirely comprised of rigid motion, burst photography [151] gives better image quality, albeit with much higher readout. (middle) Similar to current event cameras, modulated light sources trigger unwanted events that reduce bandwidth savings. However, it may be possible to ignore some of these events, perhaps by modeling the lighting variations [208].

Ego-motion events. Camera motion can trigger events in static regions, although our methods still yield substantial compression (130× over SPAD



Rigid dynamics (200× compression) | Phone screen (11× compression) | Camera motion (130× compression)

Figure 2.10: **Limitations and failure modes. (left)** Our reconstructions (yellow inset) on dynamic scenes with rigid objects can be inferior to burst photography (green inset). **(center)** Modulated light sources, such as this phone screen, can trigger a deluge of events (change points shown in the inset). **(right)** Rapid camera motion can result in an event rate divergent from scene dynamics.

outputs, Figure 2.10 (right)). We analyze the impact of ego-motion on bandwidth savings further in Section A.6. However, single-photon cameras can emulate sensor motion by integrating flux along alternate spatiotemporal trajectories [217]. We can imagine a generalized event camera that is "ego-motion compensated," by computing events along a suitable trajectory.

Photon-stream compression. SPADs generate a torrent of data—for example, 12.5 GBps for a MPixel array at 100 kHz—that can easily overwhelm data interfaces. Generalized event cameras reduce readout by around two orders of magnitude, by decoupling readout from the SPAD's frame rate and instead basing it on scene dynamics. This could pave the way for practical, high-resolution single-photon sensors.

3 Compute: Event Neural Networks

In this chapter, we present a method for modifying frame-based networks to significantly reduce inference costs on video. We start with the observation that video data is inherently repetitive. Further, this repetition occurs at multiple levels of visual complexity, from low-level pixel values to textures and high-level semantics. We leverage this multi-level redundancy by modeling changes not only in the inputs, but also in the intermediate visual features.

We propose Event Neural Networks (EvNets), in which neurons only transmit an "event" (thereby triggering downstream computation) when they have detected a sufficient change in their activation. A defining characteristic of EvNets is that each neuron has state variables that provide it with long-term memory. These state variables allow neurons to accurately track gradual, long-term changes to their activations, such as those that occur in the presence of camera motion.

We show that it is possible to transform a wide range of existing neural networks into EvNets without re-training. We demonstrate our method on state-of-the-art architectures for both high- and low-level visual processing, including pose recognition, object detection, optical flow, and image enhancement. We observe roughly an order-of-magnitude reduction in computational costs compared to conventional networks, with minimal reductions in accuracy.

3.1 Introduction

Real-world visual data is repetitive; that is, it has the property of *persistence*. For example, observe the two frames in Figure 3.1 (top). Despite being separated by one second, they appear quite similar. Human vision relies on the persistent nature of visual data to allocate limited perceptual resources. Instead of ingesting the entire scene at high resolution, the human eye points the fovea (a small region of dense receptor cells) at areas containing motion or detail [241]. This allocation of attention reduces visual processing and eye-to-brain communication.

Processing individual frames using artificial neural networks has proven to be a competitive solution for video inference [248, 274]. This paradigm leverages advances in *image* recognition (e.g., pose estimation or object detection) and processes each frame independently without considering temporal continuity, implicitly assuming that adjacent frames are statistically independent. This assumption leads to inefficient use of resources due to the repeated processing of image regions containing little or no new information.

There has been recent interest in leveraging temporal redundancy for efficient video inference. One simple solution is to skip processing image regions containing few changes in pixel values. However, such methods cannot recognize persistence in textures, patterns, or high-level semantics when it does not coincide with persistent pixel values. See Figure 3.1 (top).

Because neural networks extract a hierarchy of features from their inputs, they contain a built-in lens for detecting repetition across many levels of visual complexity. Shallow layers detect low-level patterns, and deep layers detect high-level semantics. Temporal repetition at a given level of complexity translates to persistent values at the corresponding depth in the neural hierarchy [73]. Based on this observation, we propose Event Neural Networks (EvNets), a family of neural networks in which neurons transmit (thereby

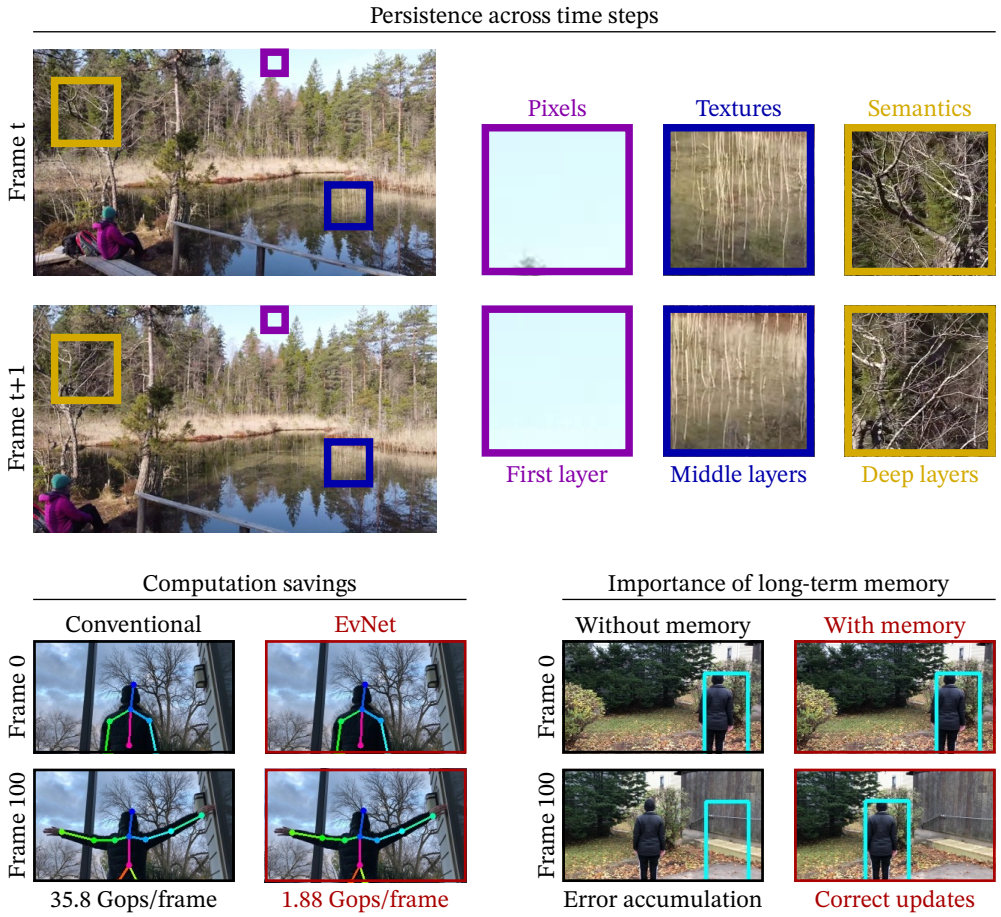


Figure 3.1: **Event Neural Networks. (top)** Two frames from a video sequence, separated by 1 second. Video source: [211]. Over this time, some areas maintain consistent pixel values (sky region). In other regions, the pixel values change but the textures (vertical lines) or semantics (tree branches) remain the same. Each type of persistence corresponds to a different depth in the neural hierarchy. EvNets leverage temporal persistence in video streams across multiple levels of complexity. **(bottom left)** EvNets yield significant computation savings while maintaining high accuracy. **(bottom right)** Event neurons have state variables that encode long-term memory, allowing EvNets to perform robust inference even over long sequences containing camera motion. A network without long-term memory (left) fails to correctly track the object due to gradual error accumulation.

triggering downstream computation) only when there is a significant change in their activation. By applying this strategy over all neurons and layers, we detect and exploit temporal persistence across many levels of complexity.

One of the defining features of EvNets is that each neuron has state variables that provide it with *long-term memory*. Instead of re-computing from scratch for every new input, an EvNet neuron accumulates information over time. Long-term memory allows EvNets to perform robust inference over long video sequences containing camera motion. See Figure 3.1 (bottom right).

We design various structural components for EvNets—both at the individual neuron level (memory state variables) and the network level (layers and transmission policies). We recognize that transmission policies, in particular, are critical for achieving a good accuracy/computation tradeoff, and we describe the policy design space in detail. We show that, with these components, it is possible to transform a broad class of conventional networks into EvNets *without re-training*. We demonstrate our methods on state-of-the-art models for several high- and low-level tasks: pose recognition, object detection, optical flow, and image enhancement. We observe roughly an order-of-magnitude reduction in arithmetic operations with minimal effects on model accuracy.

Scope and limitations. In this work, we focus on the theoretical and conceptual properties of EvNets. Although we show results on several video inference tasks, our goal is not to compete with the latest methods for these tasks in terms of accuracy. Instead, we show that, across a range of models and tasks, EvNets significantly reduce compute costs without harming accuracy.

In most of our analyses we do not assume a specific hardware platform or computation model. We mainly report arithmetic and memory operations (a platform-invariant measure of computational cost) instead of wall-clock time (which depends on many situational variables). An important next step is to consider questions relating to the design of hardware-software stacks for EvNets, in order to minimize latency and power consumption.

3.2 Related Work

Efficient neural networks. There are numerous methods for reducing the computational cost of neural networks. Many architectures have been designed to require fewer parameters and arithmetic operations [85, 93, 138, 143, 190, 270]. Another line of work uses low-precision arithmetic to achieve computation savings [44, 92, 187, 225]. Our approach is complementary to both architecture- and precision-based efficiency methods. These methods reduce the cost of inference on a single time step, whereas EvNets eliminate repetitive computation between multiple time steps. Pruning algorithms [74, 77, 122, 127] remove redundant neurons or synapses during training to improve efficiency. Instead of pruning universally redundant neurons, an EvNet adaptively ignores temporally redundant neurons.

Adaptive networks. Adaptive models modify their computation based on the input to suit the difficulty of each inference. Prior approaches consider an ensemble of sub-networks [88, 227], equip a network with multiple exits [87, 222], select the input resolution at inference time [40, 158, 257], or dynamically choose the feature resolution [246]. These methods are designed for image recognition tasks and do not explore temporal redundancy. Further, many require custom tailoring or re-training for each task and architecture. In contrast, EvNets can be readily integrated into many existing architectures and do not require re-training.

Temporal redundancy. Several recent approaches consider temporal redundancy for efficient video inference. Many take a keyframe-oriented approach, computing expensive features on keyframes, then transforming those features for the other frames [38, 98, 133, 209, 274, 276]. Other methods include using visual trackers [254], skipping redundant frames [65, 247], reusing previous frame features [159], distilling results from previous time steps [167], two-stream computation [57], and leveraging video compression [245]. In general,

these methods require extensive modifications to the network architecture.

Skip-convolution networks (Skip-Conv) [73] are closely related to EvNets. Skip-Conv reuses activation values that have not changed significantly between frames. However, the algorithm only tracks changes between consecutive frames and thus requires frequent re-initialization to maintain accuracy. Re-initialization leads to reduced efficiency, especially in the presence of camera motion. In contrast, the long-term memory in an EvNet maintains accuracy and efficiency over hundreds of frames, even when there is strong camera motion. See Figure 3.1 (bottom right).

Sigma-Delta networks [168] exploit temporal redundancy by quantizing the changes in neuron activations. Sigma-Delta networks have been limited so far to simple tasks like digit classification. Unlike Sigma-Delta networks, EvNets do not require quantization (although they do allow it). Compared to Sigma-Delta networks, EvNets achieve superior accuracy/computation tradeoffs (Figure 3.5) and generalize better to challenging, real-world tasks (Figure 3.7).

DeltaCNN [176] is concurrent work with similar goals to this work. Like EvNets, DeltaCNN has mechanisms for integrating long-term changes. They focus on translating theoretical speedups into GPU wall-time savings by enforcing structured sparsity (all channels at a given location transmit together). Despite its practical benefits, this design is inefficient when there is camera motion. In contrast, we emphasize broad conceptual frameworks (e.g., arbitrary sparsity structure) with an eye toward future hardware architectures (Section 3.6).

Event sensor inference. Event sensors [137] generate sparse frames by computing a quantized temporal gradient at each pixel. Many networks designed for inference on event-sensor data have efficient, sparse dynamics [26, 160]. However, they make strong assumptions about the mathematical properties of the network (e.g., that it is piecewise linear [160]). EvNets place far fewer

constraints on the model and are compatible an array of existing architectures.

Recurrent neural networks (RNNs). EvNets use long-term memory to track changes, and are thus loosely connected to RNNs. Long-term memory has been widely adopted in RNNs [84]. Several recent works also propose adaptive inference for RNNs by learning to skip state updates [25] or updating the state only when a significant change occurs [166, 169]. Unlike EvNets, these approaches are tailored for RNNs and generally require re-training.

3.3 Event Neurons

Consider a neuron in a conventional neural network. Let $\boldsymbol{x} = [x_1, x_2, \dots, x_n]$ be the vector of input values and y be the output. Suppose the neuron composes a linear function g (e.g., a convolution) with a nonlinear activation f. That is,

$$\begin{aligned} g(\boldsymbol{x}) &= \sum_{i=1}^{n} w_i x_i \\ y &= f(g(\boldsymbol{x})), \end{aligned} \tag{3.1}$$

where $\boldsymbol{w} = [w_1, w_2, \dots, w_n]$ contains the weights of the function g. In a conventional network, every neuron recomputes f and g for every input frame (Figure 3.2 (left)), resulting in large computational costs over a video sequence.

Inspired by prior methods that exploit persistence in activations [25, 73, 168], we describe a class of *event neurons* with *sparse*, *delta-based* transmission.

Sparse, delta-based transmission. An event neuron transmits its output to subsequent layers only when there is a sufficient change between its current activation and the previous transmission. This gating behavior makes the layer output *sparse*. However, a *value* transmission may still trigger many downstream computations (neurons receiving updated input values must recompute their activations from scratch). See Figure 3.2 (middle). Therefore,

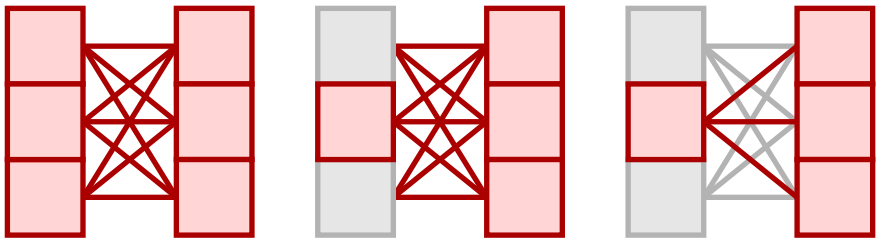


Figure 3.2: **Sparse, delta-based transmission. (left)** Conventional neurons completely recompute their activations on each time step. **(middle)** Value-based event neurons only transmit activations that have changed significantly. However, a value-based transmission can still trigger many computations. **(right)** Delta-based event neurons only transmit differential updates.

instead of transmitting an activation *value*, an event neuron transmits a *delta*.

Suppose a neuron receives a vector of incoming differentials $\boldsymbol{\Delta}_{\text{in}}$ (one element per incoming synapse). $\boldsymbol{\Delta}_{\text{in}}$ is sparse, i.e., it only contains nonzero values for upstream neurons that have transmitted. The updated g is given by

$$g(\boldsymbol{x} + \boldsymbol{\Delta}_{\text{in}}) = g(\boldsymbol{x}) + g(\boldsymbol{\Delta}_{\text{in}}). \tag{3.2}$$

Instead of computing $g(\boldsymbol{x}+\boldsymbol{\Delta}_{\text{in}})$ from scratch, an event neuron stores the value of $g(\boldsymbol{x})$ in a state variable a. When it receives a new input $\boldsymbol{\Delta}_{\text{in}}$, the neuron retrieves the old value of $g(\boldsymbol{x})$ from a, computes $g(\boldsymbol{\Delta}_{\text{in}})$, and saves the value $g(\boldsymbol{x}) + g(\boldsymbol{\Delta}_{\text{in}})$ in a. This process only requires computing products $w_i x_i$ for nonzero elements of $\boldsymbol{\Delta}_{\text{in}}$, i.e., computation scales linearly with the number of event transmissions.

The activation function f is nonlinear, so we cannot update it incrementally like g. Whenever a changes, we recompute $f(a)$, then store the updated value in another state variable. f is usually a lightweight function (e.g., ReLU), so the cost of recomputing f is far smaller than computing the products $w_i x_i$.

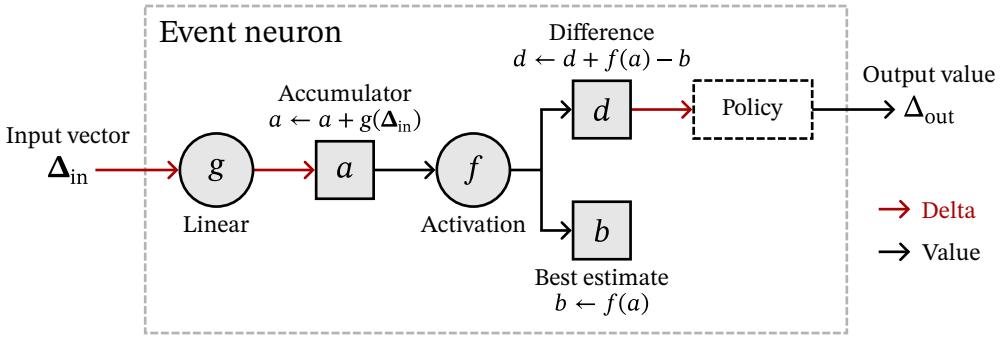


Figure 3.3: **Building event neurons.** The state variables and update rules in an event neuron. The incremental updates to a convert from a delta-based representation to value-based. The subtraction $f(a) - b$ returns the output to delta-based.

3.3.1 Building Event Neurons

An event neuron consists of three state variables, as shown in Figure 3.3. The *accumulator* (a) stores the current value of $g(\boldsymbol{x})$. The *best estimate* (b) stores the current value of $f(a)$. The *difference* (d) stores difference between b and the most recent output. When a neuron receives a differential update $\boldsymbol{\Delta}_{\text{in}}^{(t)}$ at time t from one or more of its inputs, it updates its state variables as follows:

$$\begin{aligned} a^{(t+1)} &= a^{(t)} + g(\boldsymbol{\Delta}_{\text{in}}^{(t)}) \\ d^{(t+1)} &= d^{(t)} + f(a^{(t+1)}) - b^{(t)} \\ b^{(t+1)} &= f(a^{(t+1)}). \end{aligned} \tag{3.3}$$

A neuron transmits an output Δ_{out} when some condition on d is satisfied. This condition is defined by the *transmission policy*. The transmission policy also gives the relationship between d and Δ_{out}. The policies in this work simply set $\Delta_{\text{out}} = d$. However, other relationships are possible, and the properties described in Section 3.3.2 hold for other relationships. After a neuron transmits, it sets d to $d - \Delta_{\text{out}}$. See Section 3.4.3 for more details.

3.3.2 Properties of Event Neurons

Long- and short-term memory. The state variable d accumulates all not-yet-transmitted corrections to the neuron output. It represents the neuron's *long-term memory*, whereas b represents its *short-term memory*. Including a long-term memory keeps the neuron from discarding information when it does not transmit. This *error-retention property* grants certain guarantees on the neuron's behavior, as we demonstrate next.

Error retention. Consider an event neuron receiving a series of inputs over T time steps, $\boldsymbol{\Delta}_{\text{in}}^{(1)}, \boldsymbol{\Delta}_{\text{in}}^{(2)}, \ldots, \boldsymbol{\Delta}_{\text{in}}^{(T)}$. Assume that the state variables a, b, and d have initial values $a^{(0)}$, $f(a^{(0)})$, and zero, respectively. Let the transmitted output values at each time step be $\Delta_{\text{out}}^{(1)}, \Delta_{\text{out}}^{(2)}, \ldots, \Delta_{\text{out}}^{(T)}$ (some of these may be zero). By repeatedly applying the neuron update rules, we arrive at the state

$$\begin{aligned} a^{(T)} &= a^{(0)} + g\left(\sum_{t=1}^{T} \boldsymbol{\Delta}_{\text{in}}^{(t)}\right) \\ b^{(T)} &= f(a^{(T)}) \\ d^{(T)} &= b^{(T)} - b^{(0)} - \sum_{t=1}^{T} \Delta_{\text{out}}^{(t)}. \end{aligned} \tag{3.4}$$

See Section B.5 for a detailed derivation. Observe that d is equal to the difference between the actual and transmitted changes in the activation. This is true regardless of the order or temporal distribution of the $\boldsymbol{\Delta}_{\text{in}}$ and Δ_{out}. Because the neuron stores d, it always has enough information to bring the transmitted activation into exact agreement with the true activation b. We can use this fact to bound the error within an EvNet. For example, we can constrain a neuron's error to the range $[-h, +h]$ by transmitting when $|d| > h$.

The importance of long-term memory. For comparison, consider a model in which neurons compute the difference between b on adjacent time steps, then either transmit or discard this difference without storing the remainder.

This is the model used in Skip-Conv [73]. Under this model, the final state of a neuron depends strongly on the order and temporal distribution of inputs.

For example, suppose a neuron transmits if the frame-to-frame difference exceeds a threshold δ. Consider a scenario where the neuron's activation gradually increases from 0 to 2δ in steps $0.1\delta, 0.2\delta, \ldots, 2\delta$. Gradual changes like this are common in practice (e.g., when panning over a surface with an intensity gradient). Because $0.1\delta < \delta$, the neuron never transmits and ends in a state with error -2δ. The neuron carries this error into all of its future computations. Furthermore, because the neuron discards non-transmitted activations, it has no way to know that this -2δ error exists.

3.4 Event Networks

3.4.1 Building Event Networks

So far, we have considered the design and characteristics of individual event neurons. In this section, we broaden our view and consider layers and networks. A "layer" is an atomic tensor operation (e.g., a convolution). By this definition, g and f in Section 3.3.1 correspond to two different layers.

We define three new layer types. An *accumulator layer* consists of a state vector $\boldsymbol{a}$ that contains the a variables for a collection of neurons. A *gate layer* contains state vectors $\boldsymbol{b}$ and $\boldsymbol{d}$ and the transmission policy. A *buffer layer* stores its inputs in a state vector $\boldsymbol{x}$ for future use by the next layer; this is required before non-pointwise, nonlinear layers like max pooling. The state vectors $\boldsymbol{a}$, $\boldsymbol{b}$ and $\boldsymbol{d}$ are updated using vectorized versions of the rules in Equation 3.3. An accumulator layer converts its input from delta-based to value-based, whereas a gate converts from value-based to delta-based.

To create an EvNet, we insert gates and accumulators into a pretrained network such that linear layers receive delta inputs and nonlinear layers receive value

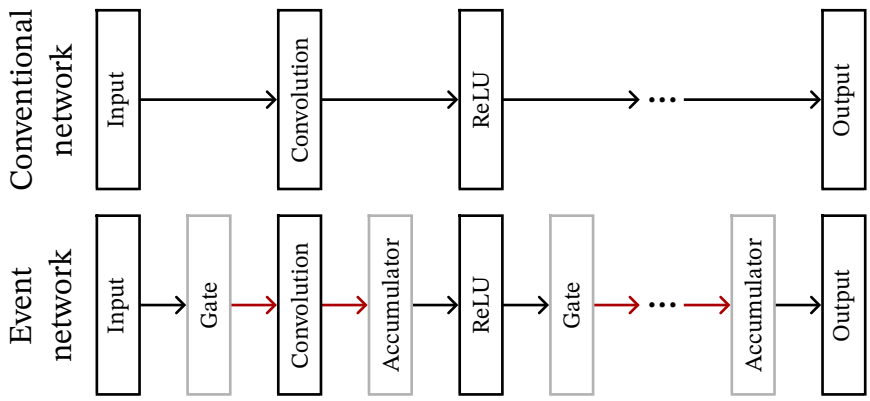


Figure 3.4: **Building event networks.** We insert accumulators and gates to make the input to linear layers (e.g., convolutions) delta-based and the input to nonlinear layers (e.g., ReLU activations) value-based.

inputs (Figure 3.4). Note that residual connections do not require any special treatment—in an EvNet, residual connections simply carry deltas instead of values. These deltas are added or concatenated to downstream deltas when the residual branch re-joins the main branch (like in a conventional network).

We place a gate at the beginning of the network and an accumulator at the end. At the input gate, we use pixel values instead of $f(a)$ and update b and d at every time step. At the output accumulator, we update $\boldsymbol{a}$ sparsely but read all its elements at every frame. Throughout the model, the functions computed by the preexisting layers (the f and g) remain the same.

3.4.2 Network Initialization

The equations in Section 3.3.1 define how to update the neuron state variables, but they do not specify those variables' initial values. Consider a simple initialization strategy where $a = 0$ and $d = 0$ for all neurons. Since the activation function f is nonlinear, the value of the state variable $b = f(a)$ may be nonzero. This nonzero b usually translates to a nonzero value of a in

the next layer. However, we initialized $a = 0$ for all neurons. Therefore, we have an inconsistency.

To address this problem, we define the notion of *internal consistency*. Consider a neuron with state variables a, d, and b. Let $\boldsymbol{b}_{\text{in}}$ and $\boldsymbol{d}_{\text{in}}$ be vectors containing the states of the neurons in the previous layer. We say that a network is in an internally consistent state if, for all neurons,

$$\begin{aligned} a &= g(\boldsymbol{b}_{\text{in}} - \boldsymbol{d}_{\text{in}}) \\ b &= f(a). \end{aligned} \tag{3.5}$$

The simplest way to satisfy these criteria is to flush some canonical input through the network. Starting with neurons in the first layer and progressively moving through all subsequent layers, we set $a = g(\boldsymbol{b}_{\text{in}})$, $b = f(a)$, and $d = 0$. In our experiments, we use the first input frame as the canonical input.

3.4.3 Transmission Policies

A *transmission policy* defines a pair of functions $M(\boldsymbol{d})$ and $P(\boldsymbol{d})$ for each layer. M outputs a binary mask $\boldsymbol{m}$ indicating which neurons should transmit. P outputs the values of $\boldsymbol{\Delta}_{\text{out}}$. In this subsection, we describe the transmission policy design space. The choice of policy is a critical design consideration, strongly influencing the accuracy and efficiency of the final model.

Locality and granularity. Policies may have different levels of *locality*, defined as the number of elements from $\boldsymbol{d}$ required to compute each element of $\boldsymbol{m}$ and $\boldsymbol{\Delta}_{\text{out}}$. A *global* policy considers all elements of $\boldsymbol{d}$ when computing each value m_i and $\Delta_{\text{out},i}$. A *local* policy considers some strict subset of $\boldsymbol{d}$, and an *isolated* policy considers only the element d_i.

In addition to its locality, each policy has a *granularity*. The granularity defines how m-values are shared between neurons. A *chunked* policy ties neurons together into local groups, producing one value of m for each group. Neurons

in the same group fire in unison. This might be practically desirable for easy parallelization on the hardware. In contrast, a *singular* policy assigns every neuron a separate value of m, so each neuron fires independently.

A linear-cost policy. In this work, we use an isolated, singular policy based on a simple threshold. Specifically,

$$\begin{aligned} m_i &= H(|d_i| - h_i) \\ \Delta_{\text{out},i} &= d_i, \end{aligned} \tag{3.6}$$

where H is the Heaviside step function and h_i is the threshold for neuron i. A key advantage of this policy is its low overhead. On receiving an incoming transmission, a neuron evaluates $|d| > h$ (one subtraction) in addition to the usual updates to a, d, and b. Neurons not receiving any updates (e.g., those in a static image region) do not incur any overhead for policy computations. In other words, the policy's cost is linear in the number of updated neurons. Combined with the linear cost of computing the neuron updates, this results in EvNets whose overall *cost scales linearly with the amount of change in the input, not with the quantity of input data received*.

This linear cost has important implications for networks processing data from high-speed sensors (e.g., event sensors [137] or single-photon sensors [53]). Here, the differences between adjacent inputs are often minuscule, and the cost of a policy with fixed per-frame overhead (e.g., a Gumbel gate [73]) could come to dominate the runtime. EvNets with a linear-overhead policy are a natural solution for processing this type of high-speed data.

Policy design and quantization. When a policy is both isolated and singular, we can characterize the functions $M(\boldsymbol{d})$ and $P(\boldsymbol{d})$ by scalar functions $M(d_i)$ and $P(d_i)$. Taking the product $M(d_i) \cdot P(d_i)$ gives a *response function* $R(d_i)$ that describes the overall behavior of the neuron. Figure 3.5 (left) illustrates several possible response functions.

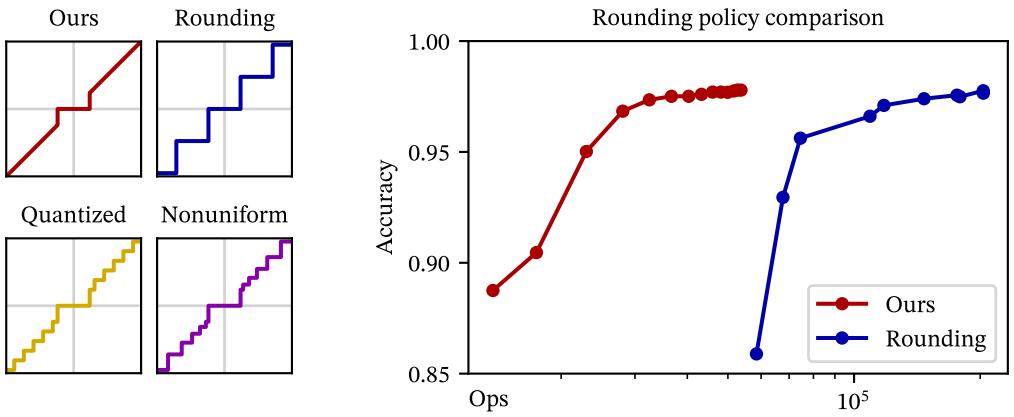


Figure 3.5: **Policy design and quantization. (left)** A few sample response functions (assuming an isolated, singular policy). **(right)** A comparison between our policy and a rounding policy (used in Sigma-Delta networks [168]). Results are for a 3-layer fully-connected network on the Temporal MNIST dataset [168].

Some response functions employ quantization to reduce the cost of computing dot product terms $w_i x_i$ (Equation 3.1). Sigma-Delta networks [168] use a rounding policy to quantize neuron outputs; a neuron transmits if this rounded value is nonzero. This rounding policy has significantly worse accuracy-computation tradeoffs (Figure 3.5 (right)) compared to our proposed policy. This might be caused by coupling the firing threshold with the quantization scale. To increase its output precision a Sigma-Delta network must reduce its firing threshold, possibly resulting in unnecessary transmissions.

3.5 Experiments

EvNets are widely applicable across architectures and video inference tasks. Any network satisfying a few basic requirements (i.e., frame-based and composing linear functions with nonlinear activations) can be converted to an EvNet *without re-training*. To demonstrate this, we select widespread, repre-

sentative models for our main experiments: YOLOv3 [190] for video object detection and OpenPose [28] for video pose estimation. Additionally, we conduct ablation experiments and report results on low-level tasks (optical flow and image enhancement).

In Appendix B, we include additional results on HRNet [216] for pose estimation. We also analyze the effect of granularity on savings, improved temporal smoothness, and provide a comparison to simple interpolation.

3.5.1 Video Pose Estimation

Dataset and experiment setup. We conduct experiments on the JHMDB dataset [100] using the widely adopted OpenPose model [28]. We use weights pre-trained on the MPII dataset [6] from [28] and evaluate the models on a subset of JHMDB with 319 videos and over 11k frames, following [73]. We report results on the combination of the three JHMDB test splits. We use the PCK metric [259] with a detection threshold of $\alpha = 0.2$, consistent with prior works [73].

Implementation details. We resize all videos to 320×240, padding as needed to preserve the aspect ratio of the content. The joint definitions in MPII (the training dataset for OpenPose) differ slightly from those in JHMDB. During evaluation, we match the JHMDB "neck," "belly," and "face" joints to the MPII "upper neck," "pelvis," and "head top" joints, respectively.

Baselines. We consider the following baselines, all using OpenPose.

- **Conventional**: The vanilla OpenPose model without modifications
- **Skip-Conv**: A variant of the Skip-Conv method with norm gates and without periodic state resets
- **Skip-Conv-8**: Adds state resets to Skip-Conv by re-flushing every 8 frames to reduce the effect of long-term activation drift

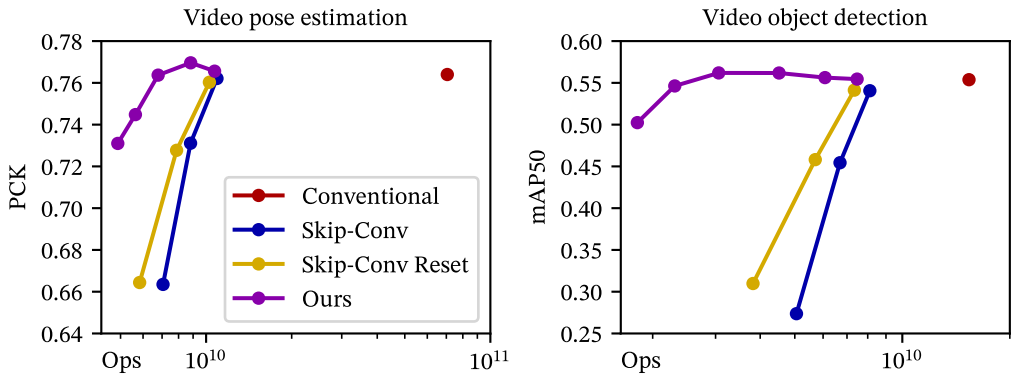


Figure 3.6: **Pareto curves.** The performance of an EvNet over several different thresholds, with baselines for comparison. The "Skip-Conv-8" model reflushes the network every 8 frames. EvNets give significant computation savings without sacrificing accuracy. See the supplement to our paper [50] for a table with this data.

We recognize that Skip-Conv networks can also incorporate a learnable gating function (the Gumbel gate) that uses information from a local window around each neuron. This can also be used for our EvNets (it is local and chunked rather than isolated and singular), but it requires re-training of the network and can incur a higher computational overhead. To keep the analysis fair, we only compare to the Skip-Conv norm gate.

Results. Figure 3.6 (a) presents our results. We vary the policy threshold h to characterize the accuracy/computation Pareto frontier. For both Skip-Conv and EvNets, increasing the threshold reduces the computational cost but increases the error rate. EvNets consistently outperform their direct competitors (Skip-Conv and Skip-Conv reset) on the Pareto frontier, achieving significantly higher accuracy when using a similar amount of computation. Surprisingly, compared to the conventional OpenPose model, EvNets sometimes have slightly better accuracy, even with a large reduction in computation. We hypothesize that this is caused by a weak inter-frame ensembling effect.

Table 3.1 summarizes the accuracy and computation at the best operating

Table 3.1: **Accuracy and computation.** Results on the best threshold for each model. We choose the highest threshold that reduces PCK or mAP by less than 0.5%.

	Pose estimation			Object detection		
Model	Thresh.	PCK (%)	Operations	Thresh.	mAP (%)	Operations
Conventional	—	**76.40**	7.055×10^{10}	—	55.38	1.537×10^{10}
Skip-Conv	0.01	76.03	1.027×10^{10}	0.01	54.13	7.340×10^{9}
Skip-Conv-8	0.01	76.21	1.092×10^{10}	0.01	54.06	8.111×10^{9}
EvNet	0.04	76.37	$\mathbf{6.780\times10^{9}}$	0.08	**56.19**	$\mathbf{3.061\times10^{9}}$

point on the Pareto curve. For each model, we choose the highest threshold that reduces PCK by less than 0.5%. To better understand the accuracy-computation tradeoff, we further report the compute and memory overhead of our EvNets (at the best operating point) in Table 3.2. We report overhead operations both as a number of operations and as a percentage. This percentage gives the ratio between "extra operations expended" and "number of arithmetic operations saved." For example, an arithmetic overhead of 0.12% indicates that the neuron updates and transmission policy require 0.12 extra arithmetic operations for every 100 operations saved. Overall, EvNets add minimal operation overhead and manageable additional memory.

3.5.2 Video Object Detection

Dataset, experiment setup, and baselines. We evaluate on the ILSVRC 2015 VID dataset [197] using the popular YOLOv3 model [190] with pre-trained weights from [199]. We report all results on the validation set with 555 videos and over 172k frames, using mean Average Precision (mAP) with an IoU threshold of 0.5 (following previous works [38, 199, 276]). We evaluate the same model variants as in Section 3.5.1 (conventional, EvNet, Skip-Conv, and Skip-Conv reset).

Implementation details. We resize all videos to 224×384, padding as needed to preserve the aspect ratio. Unlike OpenPose, YOLOv3 includes batch normal-

Table 3.2: **Overhead.** "Weights" gives the amount of memory required for model weights. "Variables" gives the amount of memory required for the state variables a, b, and d. "Arithmetic" indicates the number of extra arithmetic operations expended for neuron updates (Equation 3.3) and policy-related computations. "Load and Store" indicates the number of extra memory access operations. See the text for an explanation of the percentage notation.

		Memory costs		Operation overhead	
Model	Thresh.	Weights	Variables	Arithmetic	Load and Store
OpenPose	0.04	206.8 MB	346.2 MB	7.570×10^7 (0.12%)	1.342×10^8 (0.21%)
YOLO	0.08	248.8 MB	232.2 MB	6.417×10^7 (0.52%)	1.040×10^8 (0.85%)

ization (BN) layers. BN gives us a convenient way to estimate the distribution of activations at each neuron. We use this information to adjust the threshold values. Specifically, we scale the threshold at each neuron by $1/\gamma$ (where γ is the learned BN re-scaling parameter). This scaling makes the policy more sensitive for neurons with a narrower activation distribution, where we would expect equal-sized changes to be more meaningful.

Results. Figure 3.6 presents our results with varying thresholds. Again, we observe that our EvNets outperform Skip-Conv variants, and sometimes have slightly higher accuracy than the conventional model with greatly reduced compute cost. Table 3.1 presents the accuracy and computation at the best operating points.

3.5.3 Low-Level Vision Tasks

We have so far considered only high-level inference tasks. However, EvNets are also an effective strategy for low-level vision. We consider PWC-Net [215] for optical flow computation and HDRNet [64] for video frame enhancement. For brevity, we only show sample results in Figure 3.7 and refer the reader to Section B.1 for more details. As with the high-level models, we observe minimal degradation in accuracy and significant computation savings.

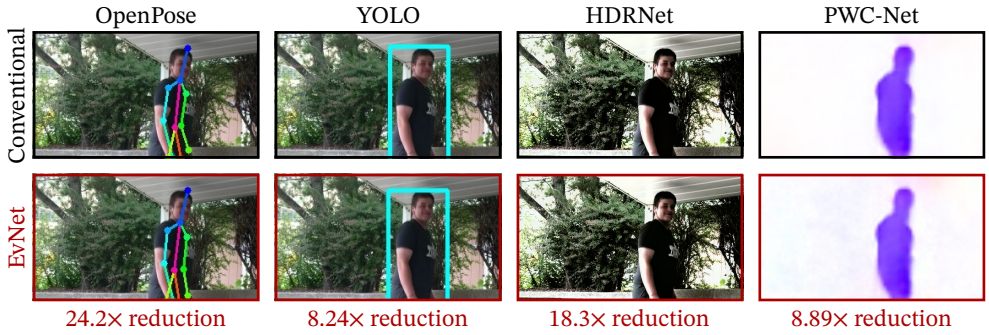


Figure 3.7: **Versatility of EvNets.** We demonstrate that EvNets are an effective strategy for many high- and low-level vision tasks. Across tasks, we see significant computation savings while maintaining high-quality output. This frame shows a person mid-jump. The EvNet tracks the subject correctly under rapid motion.

3.5.4 Ablation and Analysis

Rounding policy comparison. Figure 3.5 (right) compares our transmission policy and the rounding policy used in a Sigma-Delta network [168]. We obtain these results by evaluating the fully-connected model from the Sigma-Delta paper (with the authors' original weights) on the Temporal MNIST dataset [168]. We evaluate EvNets with thresholds of the form 10^p, where $p \in$ {-1.5, -1.4, ..., -0.3, -0.2}. We obtain results for the Sigma-Delta network using the original authors' code, which involves training the quantization threshold (the Pareto frontier is a consequence of varying a training penalty scale λ).

Ablation of long-term memory. Figure 3.8 shows the effect of ablating the long-term memory d (resetting it to zero after each input). We evaluate the OpenPose model on the JHMDB dataset. Other than resetting d, the two models shown are identical. Both models use a threshold of 0.05. We see that long-term memory is critical for maintaining stable accuracy.

Camera motion. Global camera or scene motion (e.g., camera shake or scene translation) reduces the amount of visual persistence in a video. We

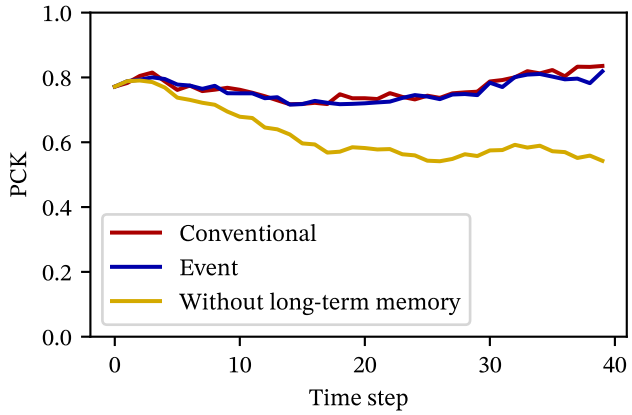


Figure 3.8: **Ablation of long-term memory.** Removing the memory d, as considered in Skip-Conv [73], causes a rapid decay in accuracy. Results using the OpenPose model on the JHMDB dataset. See Section 3.5 for details.

would therefore expect camera motion to reduce the savings in an EvNet. To confirm this, we evaluate the OpenPose and YOLO models on a custom-labeled video dataset. We label the camera motion in each video as "none" (perfectly stationary camera), "minor" (slight camera shake), or "major." See Section B.4 for details. We test OpenPose with a threshold of 0.05 and YOLO with a threshold of 0.06. Because this dataset does not have frame-level labels for pose or object detection, we do not explicitly evaluate task accuracy. However, the thresholds we use here give good accuracy on JHMDB and VID. For OpenPose, the computation savings for "none," "major," and "minor" camera motion are 17.3×, 11.3×, and 8.40×, respectively. For YOLO, the savings are 6.64×, 3.95×, and 2.65×. As expected, we see a reduction in savings when there is strong camera motion, although we still achieve large reductions relative to the conventional model.

Wall-time savings. We now show preliminary results demonstrating wall-time savings in EvNets. We consider the HRNet [216] model (see Section B.3) on the JHMDB dataset. We evaluate on an Intel Core i7 8700K CPU.

We implement the model in PyTorch. For the EvNet, we replace the standard convolution with a custom sparse C++ convolution. Our convolution uses an input-stationary design (i.e., an outer loop over input pixels) to skip zero deltas efficiently. In the conventional model, we use a custom C++ convolution with a standard output-stationary design (i.e., an outer loop over output pixels). We use a custom operator in the conventional model to ensure a fair comparison, given the substantial engineering effort invested in the default MKL-DNN library. We implement both operators with standard best practices (e.g., maximizing data-access locality). We compile with GCC 9.4 with the `-Ofast` flag.

For evaluation, we use an input size of 256×256 and an EvNet threshold of 0.1. The EvNet achieves a PCK of 90.46% and runs in an average of 0.3497 s (7.361×10^8 ops) per frame. The conventional model achieves a PCK of 90.37% and runs in 1.952 s (1.019×10^{10} ops) per frame.

Layer trends. Figure 3.9 shows the computational cost of the OpenPose model as a function of the layer depth. We show results both on the JHMDB dataset and on our custom-labelled MPII dataset (to allow analysis of the effect of camera motion). Overall, we see a reduction in the relative cost as we go deeper in the network. This highlights the importance of leveraging repetition in the deep layers of the network, not just near the input. We also see that early layers transmit more often when there is camera motion. This corresponds to more changes in low-level features and pixel values.

Temporal variation. Figure 3.10 shows the per-frame computational cost of the OpenPose EvNet over the course of a video. The video in question has a static background and a moving foreground object (person). Recognizable events in the video (e.g., walking, jumping) correspond to temporary increases in the number of operations. In this way, we see EvNets living up to their promise of "only computing when something interesting is happening."

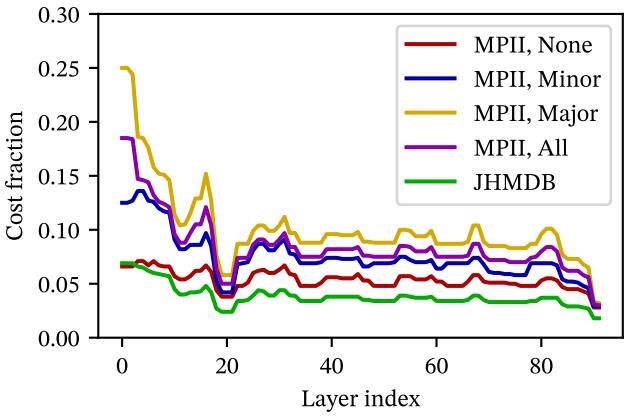


Figure 3.9: **Operation costs by layer.** Results for the OpenPose model on the JHMDB and custom-labelled MPII datasets. The increasing savings with depth show the importance of leveraging repetition at all levels of the network hierarchy. We have applied a median filter of size 5 (along the layer axis) to the data in this plot.

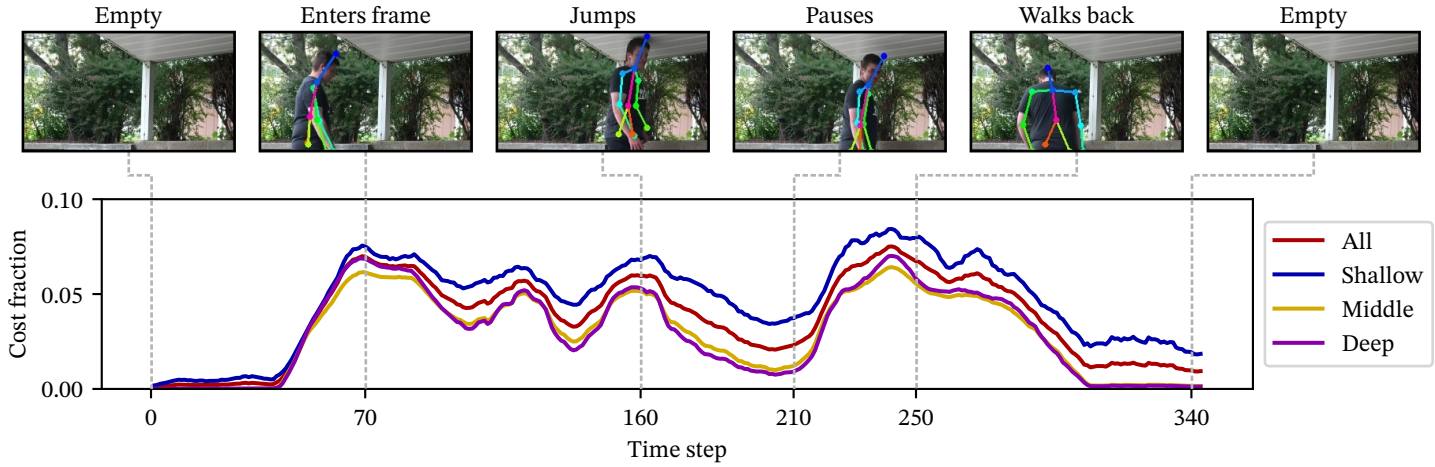


Figure 3.10: **Temporal variation in operation cost.** Identifiable events in the video (e.g., jumping) correspond to temporary increases in the number of operations. "Shallow" corresponds to the first 31 layers, "middle" to the next 31, and "deep" to the final 30. We have applied a centered moving average of size 10 (along the time axis) to the data in this plot.

3.6 Discussion

Hardware platforms. Mainstream GPU hardware is designed for parallel, block-wise computation with coarse control flow. EvNets with neuron-level transmission are inefficient under this computation model. In the long term, we expect to achieve the best performance on specialized hardware designed for extreme parallelism and distributed control. It is important to emphasize that event neurons *do not need to operate by a shared clock*. Each neuron operates independently—consuming new input as it arrives and transmitting output once it is computed. This independence permits an asynchronous, networked execution model in contrast to the ordered, frame-based model in conventional machine learning. Spiking neural networks (SNNs) [153] share this asynchronous computation model and have motivated the development of neuromorphic hardware platforms [3, 46] that could be re-purposed for efficient implementation of EvNets.

4 Compute: Eventful Transformers

In this chapter, we extend the idea of event-based inference beyond CNNs to another widely-used architecture: vision Transformers. Vision Transformers achieve impressive accuracy across a range of visual recognition tasks. Unfortunately, their accuracy frequently comes with high computational costs. This is a particular issue in video recognition, where a Transformer may be applied repeatedly across frames or temporal chunks.

Similar to Chapter 3, we exploit temporal redundancy between subsequent inputs to reduce the cost of video processing. However, there are significant differences between CNNs and Transformers that require rethinking some of our previous techniques. Specifically, the self-attention operator does not satisfy the linearity assumption in Chapter 3; to address this challenge, we propose a novel method for event-based self-attention updates. This approach allows us to re-process only tokens that have changed significantly over time.

Our proposed family of models, Eventful Transformers, can be converted from existing Transformers (often without any re-training) and give adaptive control over the compute cost at runtime. We evaluate our method on large-scale datasets for video object detection (ImageNet VID) and action recognition (EPIC-Kitchens 100). Our approach leads to significant computational savings (on the order of 2–4×) with only minor reductions in accuracy.

4.1 Introduction

Transformers, initially designed for language modeling [226], have been recently explored as an architecture for vision tasks. Vision Transformers [48] have achieved impressive accuracy across a range of visual recognition problems, attaining state-of-the-art performance in tasks including image classification [48], video classification [8, 15, 54], and detection [29, 131, 146, 234].

One of the primary drawbacks of vision Transformers is their high computational cost. Whereas typical convolutional networks (CNNs) consume tens of GFlops per image [27], vision Transformers often require an order of magnitude more computation, up to hundreds of GFlops per image. In video processing, the large volume of data further amplifies these costs. High compute costs preclude vision Transformers from deployment on resource-constrained or latency-critical devices, limiting the scope of this otherwise exciting technology. In this work, we present one of the first methods to use *temporal redundancy between subsequent inputs* to reduce the cost of vision Transformers when applied to video data.

Temporal redundancy. Consider a vision Transformer that is applied frame-by-frame or clip-by-clip to a video sequence. This Transformer might be a simple frame-wise model (e.g., an object detector) or an intermediate step in some spatiotemporal model (e.g., the first stage of the factorized model from [8]). Unlike in language processing, where one Transformer input represents a complete sequence, we consider Transformers applied to several distinct inputs (frames or clips) over time.

Natural videos contain significant temporal redundancy, with only slight differences between subsequent frames. Despite this fact, deep networks (including Transformers) are commonly computed "from scratch" on each frame. This approach is wasteful, discarding all potentially relevant information from previous inferences. Our key intuition is that we can reuse

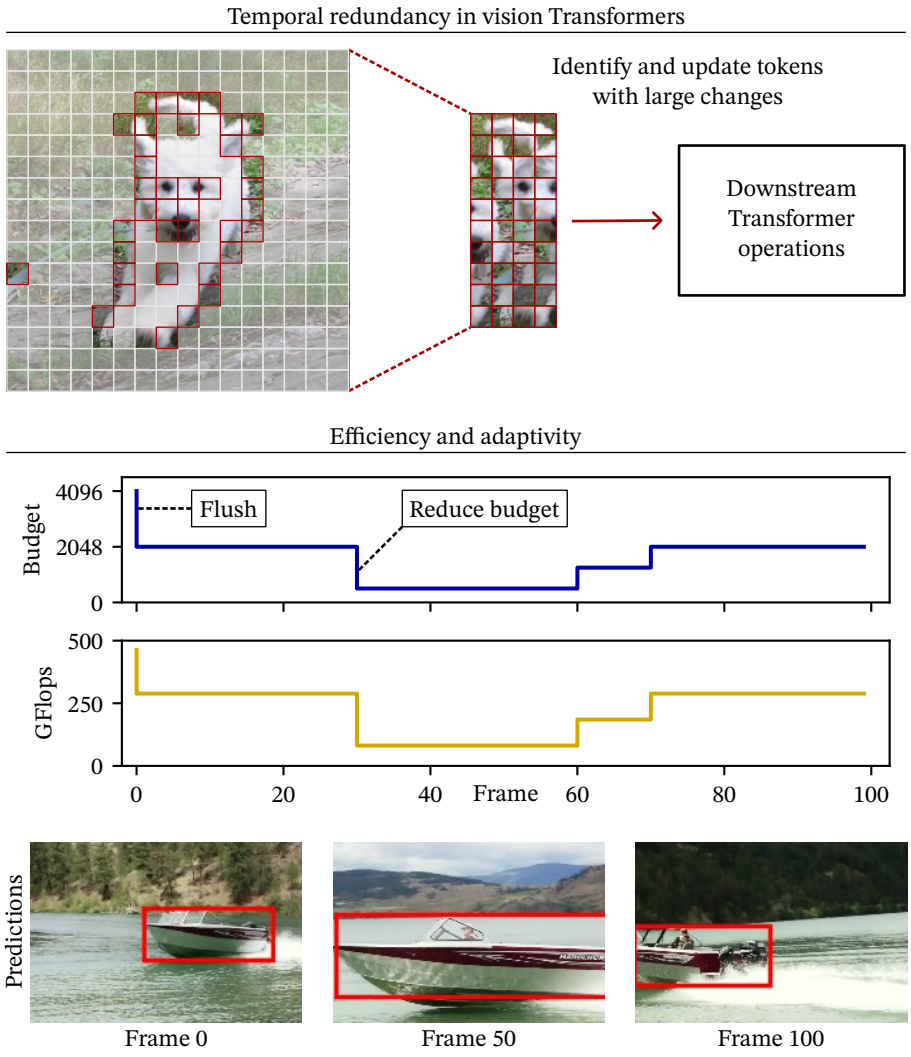


Figure 4.1: **Eventful Transformers.** Our method exploits temporal redundancy between subsequent model inputs. **(top)** Within each Transformer block, we identify and update only those tokens with significant changes over time. Image: [23]. **(bottom)** In addition to improving efficiency, our method gives fine-grained control over the compute cost at runtime. "Budget" refers to parameter r as described in Section 4.4.3. "Flush" refers to the initialization of all tokens on the first time step. This example shows the ViTDet [131] object detection model on a video from the VID [197] dataset.

intermediate computations from earlier time steps to improve efficiency on redundant sequences.

Adaptive inference. For vision Transformers (and deep networks in general), the inference cost is typically fixed by the architecture. However, in real-world applications, the available resources may vary over time (e.g., due to competing processes or variations in power supply). As such, there is a need for models whose computational cost can be modified at runtime [161]. In this work, adaptivity is one of our primary design objectives; we design our method to allow real-time control over the compute cost. See Figure 4.1 (bottom portion) for an example where we vary the compute budget throughout a video.

Challenges and opportunities. There are past works exploring temporal redundancy [50, 73, 176] and adaptivity [158, 227, 253] for CNNs. However, these methods are generally incompatible with vision Transformers, owing to substantial architectural differences between Transformers and CNNs. Specifically, Transformers introduce a new primitive, self-attention, that does not conform to the assumptions of many CNN-based methods.

Despite this challenge, vision Transformers also represent a unique opportunity. In CNNs, it is difficult to translate sparsity improvements (i.e., the sparsity gained by considering temporal redundancy) into concrete speedups. Doing so requires imposing significant restrictions on the sparsity structure [73] or using custom compute kernels [176]. In contrast, the structure of Transformer operations (centered on manipulating token vectors) makes it easier to translate sparsity into reduced runtime using standard operators.

Eventful Transformers. We propose Eventful Transformers, a new class of Transformer that leverages temporal redundancy between inputs to enable efficient, adaptive inference. The term "Eventful" is inspired by event cameras [20, 137], sensors that produce sparse outputs based on scene changes. Eventful Transformers track token-level changes over time, selectively updating the token representations and self-attention maps on each time step.

Blocks in an Eventful Transformer include gating modules that allow controlling the number of updated tokens at runtime.

Our method can be applied to off-the-shelf models (generally without retraining) and is compatible with a wide range of video processing tasks. Our experiments demonstrate that Eventful Transformers, converted from existing state-of-the-art models, significantly reduce computational costs while largely preserving the original model's accuracy. We publicly release our code, which includes PyTorch modules for building Eventful Transformers.

Limitations. We demonstrate wall-time speedups on both the CPU and GPU. However, our implementation (based on vanilla PyTorch operators) is likely sub-optimal from an engineering standpoint. With additional effort to reduce overhead (e.g., implementing a fused CUDA kernel for our gating logic), we are confident that the speedup ratios could be further improved. Our method also involves some unavoidable memory overheads. Perhaps unsurprisingly, reusing computation from previous time steps requires maintaining some tensors in memory. These memory overheads are relatively modest; see Section 4.6 for further discussion.

4.2 Related Work

Efficient Transformers. Several past works improve the efficiency of Transformers. Many of these methods focus on reducing the quadratic complexity of self-attention, often using low-rank or sparse approximations [39, 41, 70, 99, 105, 116, 149, 191, 195, 233]. We consider standard self-attention (with windowing in some cases). Our approach is orthogonal to the above methods.

Selecting and summarizing tokens. Some recent works improve the efficiency of vision Transformers by exploiting spatial redundancy within each input. Many of these methods prune or fuse tokens based on a salience measure [56, 67, 135, 170, 185]. A notable example is the Adaptive Token Sampling

(ATS) algorithm [56], which has an adaptive computation cost and does not require re-training. Other spatial redundancy methods include adaptive token pooling [17, 156], hierarchical pooling [173], learned tokenization [198], and progressive token sampling [265].

Unlike these works, which consider *spatial* redundancy, our method targets *temporal* redundancy. This makes our work complementary to these approaches. A single model can leverage both spatial and temporal redundancy by only updating tokens that are both salient *and* not temporally repetitive. We illustrate this compatibility in our experiments by building a simple proof-of-concept model that combines temporal and spatial redundancy.

Another related work is Spatiotemporal Token Selection (STTS) [231], which exploits spatiotemporal redundancy for video inference. STTS is intended for models with explicit temporal reasoning that take an entire video as input. In contrast, our method is designed for models that are repetitively applied to frames or clips. Compared to STTS, our method covers a wider range of architectures and tasks.

Temporal redundancy between inputs. There has been recent work on exploiting inter-frame temporal redundancy in CNNs [30, 50, 73, 176]. While we draw some inspiration from these methods, directly applying them to vision Transformers is not feasible due to significant architectural differences between CNNs and Transformers.

There is limited existing research on exploiting temporal redundancy between subsequent vision Transformers inputs. To our knowledge, the only past work in this area is the Spatiotemporal Gated Transformers (STGT) method [132]. There are two noteworthy differences between STGT and our work. Most notably, STGT only considers temporal redundancy within token-level operations (e.g., token-wise linear transforms), and not within the self-attention operator. Our method accelerates all major Transformer components, including self-attention. Further, STGT uses lossy gating logic that leads to accuracy

degradation on long sequences with gradual changes. Our method avoids this issue by employing an improved, reference-based gating mechanism.

Adaptive neural networks. Many existing methods add adaptivity to deep CNNs [40, 58, 87, 158, 222, 227, 236, 246, 253, 257]. However, due to architectural differences (e.g., the use of relative position embeddings in Transformers), these methods (e.g., those based on input resizing) often do not translate to vision Transformers.

There has been some recent work on adaptive vision Transformers [157, 239, 262]. These works leverage redundancy within a single input, whereas we consider redundancy between inputs. Unlike our method, these approaches generally require re-training or fine-tuning the model.

Efficient neural networks. There is a substantial body of work on improving the efficiency of deep networks. Some works propose efficient CNN architectures [85, 93, 270]. Others use reduced-precision arithmetic [44, 92, 187] or pruning [74, 77, 122, 127]. Our method is loosely connected to pruning; it can be viewed as adaptively pruning redundant tokens on each time step.

4.3 Background: Vision Transformers

In this section, we describe the basic elements of a vision Transformer (see [48] for more details) and define the mathematical notation we use throughout the rest of the chapter.

A vision Transformer consists of a sequence of Transformer blocks. The input to each block is a list of N, D-dimensional token vectors; we denote this as $\boldsymbol{x} \in \mathbb{R}^{N \times D}$. Before the first Transformer block, a vision Transformer maps each image patch to a token vector using a linear transform. Positional embedding [226] can be injected before the first block [48] or at every block [131].

A Transformer block. A Transformer block maps input $\boldsymbol{x} \in \mathbb{R}^{N \times D}$ to output

$\boldsymbol{z} \in \mathbb{R}^{N \times D}$, according to

$$\boldsymbol{y} = \mathrm{MSA}(\mathrm{LN}(\boldsymbol{x})) + \boldsymbol{x}, \tag{4.1}$$

$$\boldsymbol{z} = \mathrm{MLP}(\mathrm{LN}(\boldsymbol{y})) + \boldsymbol{y}, \tag{4.2}$$

where "MSA" denotes multi-headed self-attention. "MLP" is a token-wise multilayer perceptron with two layers and one GELU nonlinearity. "LN" denotes layer normalization.

Multi-headed self-attention (MSA). The self-attention operator first applies three linear transforms $W_q, W_k, W_v \in \mathbb{R}^{D \times D}$ to its input $\boldsymbol{x}' = \mathrm{LN}(\boldsymbol{x})$.

$$\boldsymbol{q} = \boldsymbol{x}' W_q \qquad \boldsymbol{k} = \boldsymbol{x}' W_k \qquad \boldsymbol{v} = \boldsymbol{x}' W_v. \tag{4.3}$$

$\boldsymbol{q}$, $\boldsymbol{k}$, and $\boldsymbol{v}$ are the "query," "key," and "value" tensors, respectively. In practice, W_q, W_k, W_v are often fused into a single transform $W_{qkv} = [W_q, W_k, W_v]$. These transforms may include a bias; we omit the bias here for brevity.

The self-attention operator then computes a normalized similarity matrix (attention matrix) $A \in \mathbb{R}^{N \times N}$ between the tokens of $\boldsymbol{q}$ and $\boldsymbol{k}$.

$$A = \mathrm{Softmax}\left(\boldsymbol{q}\boldsymbol{k}^T / \sqrt{D}\right). \tag{4.4}$$

Softmax normalization is applied along rows of the matrix.

The MSA output $\boldsymbol{y}'$ is an attention-weighted sum of the value tokens $\boldsymbol{v}$, followed by a linear projection W_p.

$$\boldsymbol{y}' = (A\boldsymbol{v})\, W_p. \tag{4.5}$$

Multi-headed self-attention (as opposed to single-headed self-attention) splits $\boldsymbol{q}$, $\boldsymbol{k}$, and $\boldsymbol{v}$ into H tensors of shape $\mathbb{R}^{N \times (D/H)}$ and applies self-attention in parallel across these H heads. Before applying W_p, the results of all heads are

concatenated into a tensor with shape $\mathbb{R}^{N \times D}$.

Windowed attention. Standard MSA has a complexity of $\mathcal{O}(N^2)$ (quadratic in the number of tokens). To reduce this cost, many vision Transformers adopt *windowed* attention. Windowed attention constrains attention to local windows. Information can be exchanged between windows by shifting the windows between blocks [146] or by interleaving global attention [131].

4.4 Eventful Transformers

Our goal is to accelerate vision Transformers for video recognition, in the situation where a Transformer is applied repetitively across frames or chunks of frames (e.g., for video object detection or video action recognition, respectively). Our key idea is to exploit temporal redundancy by re-using computation from previous time steps. In this section, we describe how to modify Transformer blocks to add temporal redundancy awareness.

In Section 4.4.1, we present a token-gating module that monitors temporal changes and determines which tokens to update. In Section 4.4.2, we integrate our token gating logic into a Transformer block, creating a redundancy-aware Eventful Transformer block. In Section 4.4.3, we explore policies for selecting which tokens to update.

4.4.1 Token Gating: Detecting Redundancy

In this subsection, we propose two modules: token gates and token buffers. These modules allow us to identify and update only those tokens that have changed significantly since their last update.

Gate module. A gate selects $M \leq N$ of its input tokens to send to downstream layers for re-computation. The gate maintains a set of *reference tokens* in memory, which we denote as $\boldsymbol{u} \in \mathbb{R}^{N \times D}$. The reference tensor contains the

value of each token on the time step it was most recently updated. On each time step, tokens are compared against their references; those that deviate significantly from their reference are selected for an update.

Let $\boldsymbol{c} \in \mathbb{R}^{N \times D}$ denote the current input to the gate. On each time step, we update the gate's state and determine its output according to the following procedure (see Figure 4.2):

1. Compute the total error $\boldsymbol{e} = \boldsymbol{u} - \boldsymbol{c}$.
2. Apply a *selection policy* to the error $\boldsymbol{e}$. A selection policy returns a binary mask $\boldsymbol{m}$ (equivalently, a list of token indices) indicating which M tokens should be updated.
3. Extract the tokens selected by the policy. In Figure 4.2, we depict this as the product $\boldsymbol{c} \times \boldsymbol{m}$; in practice, we achieve this with a "gather" operation along the first axis of $\boldsymbol{c}$. We denote the gathered tokens as $\tilde{\boldsymbol{c}} \in \mathbb{R}^{M \times D}$. The gate returns $\tilde{\boldsymbol{c}}$ as its output.
4. Update the references for selected tokens. In Figure 4.2, we depict this as $\boldsymbol{u} \leftarrow \boldsymbol{e} \times (\sim \boldsymbol{m}) + \boldsymbol{c} \times \boldsymbol{m}$; in practice, we apply a "scatter" operation from $\tilde{\boldsymbol{c}}$ into $\boldsymbol{u}$.

On the first time step, the gate updates all tokens (initializing $\boldsymbol{u} \leftarrow \boldsymbol{c}$ and returning $\tilde{\boldsymbol{c}} = \boldsymbol{c}$).

Buffer module. A buffer module maintains a state tensor $\boldsymbol{b} \in \mathbb{R}^{N \times D}$ that tracks the most recent known value for each of its input tokens. When receiving a new input $f(\tilde{\boldsymbol{c}}) \in \mathbb{R}^{M \times D}$, the buffer scatters the tokens from $f(\tilde{\boldsymbol{c}})$ into their corresponding locations in $\boldsymbol{b}$. It then returns the updated $\boldsymbol{b}$ as its output. See Figure 4.3.

We pair each gate with a subsequent buffer. One simple usage pattern is as follows. The gate output $\tilde{\boldsymbol{c}} \in \mathbb{R}^{M \times D}$ is passed to a series of token-wise

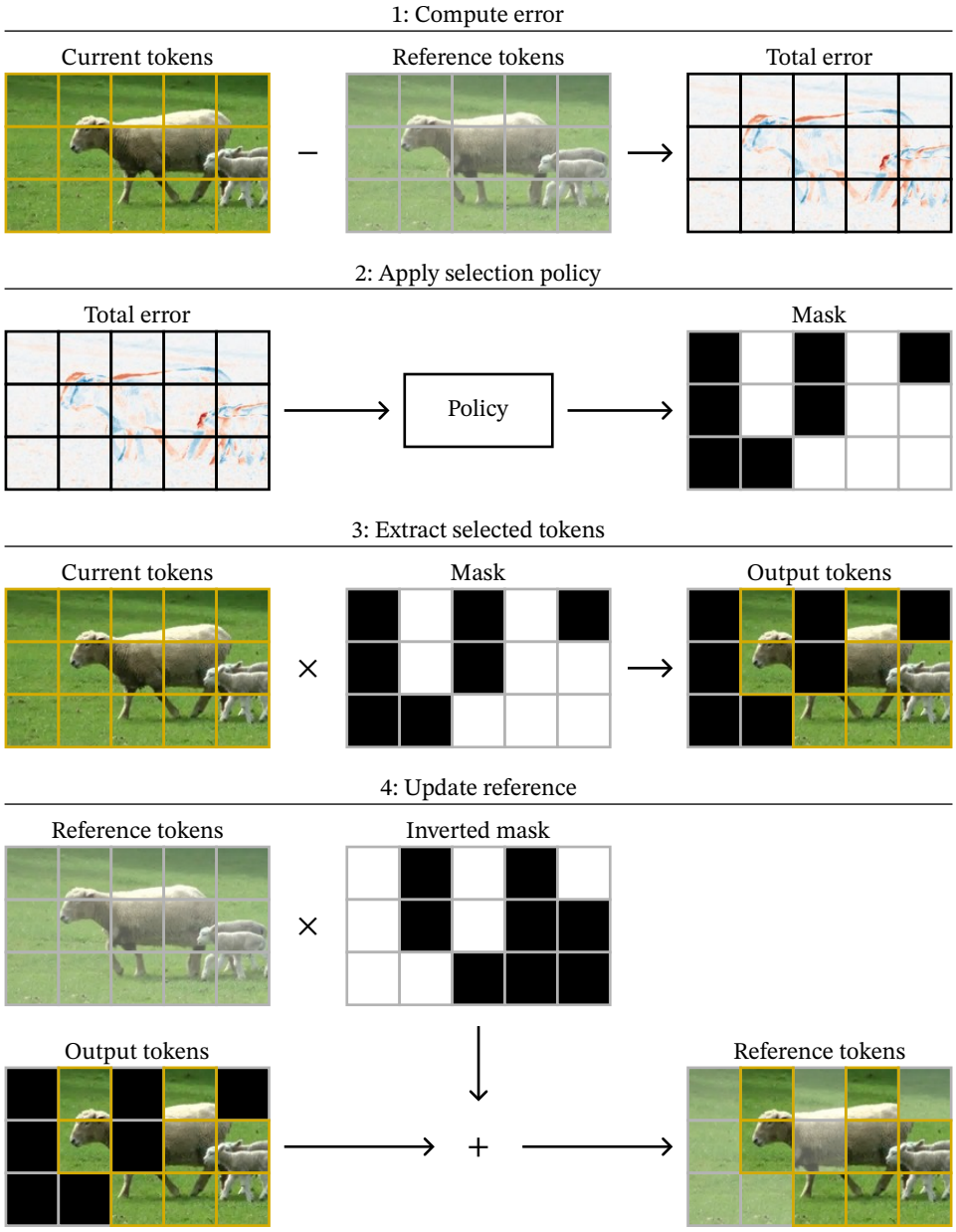


Figure 4.2: **Token gating.** A gating module compares incoming tokens against a stored reference. If the difference between a token and its reference is large, then the token is selected to be updated. See Section 4.4.3 for details on selection policies. Images are from the VID [197] dataset.

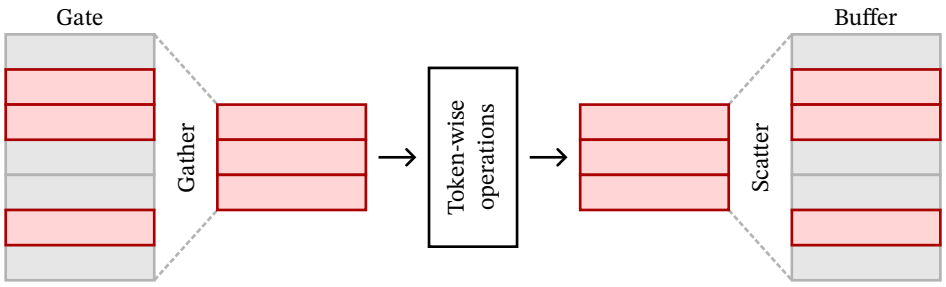


Figure 4.3: **Accelerating token-wise operations.** The gate reduces the number of active tokens from N to M. Subsequent token-wise operations operate on a smaller tensor and therefore have a lower computational cost (proportional to M).

operations $f(\tilde{\boldsymbol{c}})$. The resulting tensor $f(\tilde{\boldsymbol{c}}) \in \mathbb{R}^{M \times D}$ is then passed to a buffer, which restores the full shape $\mathbb{R}^{N \times D}$.

4.4.2 Building Redundancy-Aware Transformers

In this subsection, we propose a modified Transformer block that exploits temporal redundancy. Figure 4.4 shows our design for an Eventful Transformer block. Our method accelerates token-wise operations (e.g., the MLP), as well as the query-key and attention-value multiplications (Equations 4.4 and 4.5).

Token-wise operations. Many of the operations in a Transformer block are token-wise, meaning they do not involve information exchange between tokens. These include the MLP and the linear transforms in the MSA. We can save computation in token-wise operations by skipping those tokens not selected by the gate. Due to token-wise independence, this does not change the result of the operation for the selected tokens. See Figure 4.3.

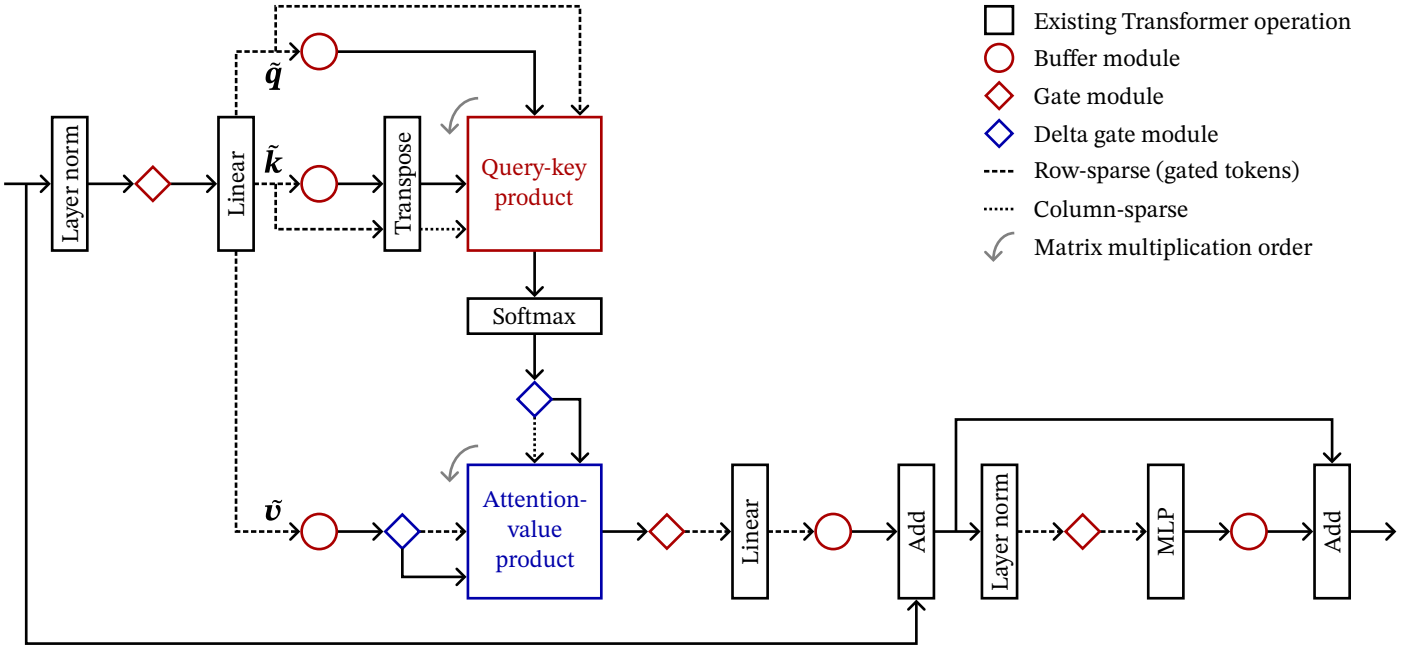


Figure 4.4: **An Eventful Transformer block.** To exploit temporal redundancy, we strategically apply token gating throughout the block and compute a modified, sparse self-attention update. Rectangles are standard Transformer components (see Section 4.3). For clarity, we have omitted some minor operations (e.g., scaling after the first matrix multiplication) from this figure.

Specifically, we place a gate-buffer pair around each contiguous sequence of token-wise operations, including the W_{qkv} transform (Equation 4.3), the W_p transform (Equation 4.5), and the MLP. Note that we add buffers before the skip connections (Equations 4.1 and 4.2) to ensure that the tokens of the two addition operands are correctly aligned.

The cost of a token-wise operation is proportional to the number of tokens. A gate reduces the number of tokens from N to M. This, in turn, reduces the computational cost of downstream token-wise operations by a factor of N/M.

The query-key product. We now consider the query-key product $B = \boldsymbol{q}\boldsymbol{k}^T$ (part of Equation 4.4). Writing this matrix multiplication explicitly, we have

$$B_{ij} = \sum_p \boldsymbol{q}_{ip}\left(\boldsymbol{k}^T\right)_{pj}. \tag{4.6}$$

Element B_{ij} needs to be updated if either (a) there is a change in the i^{th} row of $\boldsymbol{q}$, or (b) there is a change in the j^{th} column of $\boldsymbol{k}^T$. Due to the gate that we inserted before the W_{qkv} transform (shown in Figure 4.4), only some rows of $\boldsymbol{q}$ and some columns of $\boldsymbol{k}^T$ have changed. Therefore, we only need to recompute a subset of the elements of B.

Let $\tilde{\boldsymbol{x}}' \in \mathbb{R}^{M\times D}$ denote the output of the gate before the W_{qkv} transform. We define $\tilde{\boldsymbol{q}} = \tilde{\boldsymbol{x}}' W_q$ and $\tilde{\boldsymbol{k}} = \tilde{\boldsymbol{x}}' W_k$ (following Equation 4.3). Let $\boldsymbol{q}$ and $\boldsymbol{k}$ denote the outputs of the $\tilde{\boldsymbol{q}}$ and $\tilde{\boldsymbol{k}}$ buffers (shown in Figure 4.4). $\tilde{\boldsymbol{q}}$ contain $\tilde{\boldsymbol{k}}$ the subset of tokens from $\boldsymbol{q}$ and $\boldsymbol{k}$ that are being updated.

Figure 4.5 depicts our method for sparsely updating B. The product $\tilde{\boldsymbol{q}}\boldsymbol{k}^T$ contains the elements of B that need to be updated due to a change in $\tilde{\boldsymbol{q}}$. We compute $\tilde{\boldsymbol{q}}\boldsymbol{k}^T$, then scatter the result row-wise into the old B (the value of B from the last time step). We use an analogous approach for the $\tilde{\boldsymbol{k}}$-induced updates; we compute $\boldsymbol{q}\tilde{\boldsymbol{k}}^T$ and scatter the result column-wise into B.

The overall cost of these updates is $2NMD$, compared to a cost of N^2D to

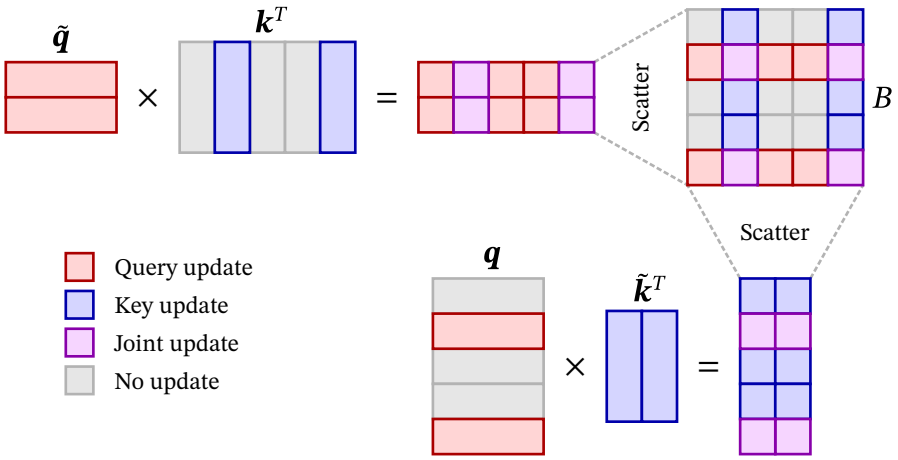


Figure 4.5: **The query-key product.** We reduce the cost of computing $B = \boldsymbol{q}\boldsymbol{k}^T$ by only updating a subset of its elements. We first compute changes induced by updated rows in $\boldsymbol{q}$ (top-left), then compute changes induced by updated columns in $\boldsymbol{k}^T$ (bottom).

compute B from scratch. Note that the cost of our method is proportional to M, the number of tokens selected by the gate. We save computation when $M < N/2$ (when we update fewer than half of the tokens).

The above method for updating B involves some redundant computation. Some elements of the first scattered matrix $\tilde{\boldsymbol{q}}\boldsymbol{k}^T$ are also present in the second matrix $\boldsymbol{q}\tilde{\boldsymbol{k}}^T$. These overlapping elements are computed twice. Eliminating this redundancy would reduce the cost of our method to $NMD \leq N^2D$. This could be achieved by removing the tokens in $\tilde{\boldsymbol{q}}$ from $\boldsymbol{q}$ before computing $\boldsymbol{q}\tilde{\boldsymbol{k}}^T$. We would then scatter the result by indexing along both axes of B. We leave this as an optimization for future implementations.

The attention-value product. We now describe a method for updating the attention-value product $A\boldsymbol{v}$ (part of Equation 4.5). Writing this multiplication explicitly, we have

$$(A\boldsymbol{v})_{ij} = \sum_p A_{ip}\boldsymbol{v}_{pj}. \tag{4.7}$$

Because of the gate before the W_{qkv} transform, only some rows (tokens) of $\boldsymbol{v}$ change on each time step. However, there are some updated values in every *column* of $\boldsymbol{v}$. Therefore, every element of $A\boldsymbol{v}$ will change on each time step. This means we cannot use the same strategy that we used for B, where we only updated some of the output elements.

Instead, we propose a delta-based update strategy. Let A_o and $\boldsymbol{v}_o$ denote the last known values for A and $\boldsymbol{v}$. Let A_Δ and $\boldsymbol{v}_\Delta$ denote changes in A and $\boldsymbol{v}$. Define $A_n = A_o + A_\Delta$ and $\boldsymbol{v}_n = \boldsymbol{v}_o + \boldsymbol{v}_\Delta$. We can compute the updated attention-value product $A_n\boldsymbol{v}_n$ as

$$\begin{aligned} A_n\boldsymbol{v}_n &= (A_o + A_\Delta)(\boldsymbol{v}_o + \boldsymbol{v}_\Delta) \\ &= A_o\boldsymbol{v}_o + A_o\boldsymbol{v}_\Delta + A_\Delta\boldsymbol{v}_o + A_\Delta\boldsymbol{v}_\Delta \\ &= A_o\boldsymbol{v}_o + (A_o + A_\Delta)\boldsymbol{v}_\Delta + A_\Delta(\boldsymbol{v}_o + \boldsymbol{v}_\Delta) - A_\Delta\boldsymbol{v}_\Delta \\ &= A_o\boldsymbol{v}_o + A_n\boldsymbol{v}_\Delta + A_\Delta\boldsymbol{v}_n - A_\Delta\boldsymbol{v}_\Delta. \end{aligned} \tag{4.8}$$

Therefore, on each time step, we can update $A\boldsymbol{v}$ by adding $A_n\boldsymbol{v}_\Delta + A_\Delta\boldsymbol{v}_n - A_\Delta\boldsymbol{v}_\Delta$ to the previous result $A_o\boldsymbol{v}_o$.

We obtain A_Δ, $\boldsymbol{v}_\Delta$, A_o, and $\boldsymbol{v}_o$ using *delta gate modules*. Delta gates are similar to the gates defined in Section 4.4.1, with one difference: instead of returning $\tilde{\boldsymbol{c}}$, a delta gate returns $\boldsymbol{u}$ and $\tilde{\boldsymbol{e}}$ (where $\tilde{\boldsymbol{e}}$ is the result of gathering the selected indices from $\boldsymbol{e}$). $\boldsymbol{u}$ represents the effective current value of the gate's output, corresponding to A_n or $\boldsymbol{v}_n$ in Equation 4.8. $\tilde{\boldsymbol{e}}$ represents the amount of change on the current time step, corresponding to A_Δ or $\boldsymbol{v}_\Delta$ in Equation 4.8.

Figure 4.6 illustrates our approach for efficiently computing the three delta terms in Equation 4.8. We remove the columns of A_n that correspond to zero rows in $\boldsymbol{v}_\Delta$ (these columns will always be multiplied by zero). Let $\tilde{A}_n$ denote A_n with these columns removed. We remove rows of $\boldsymbol{v}_n$ analogously

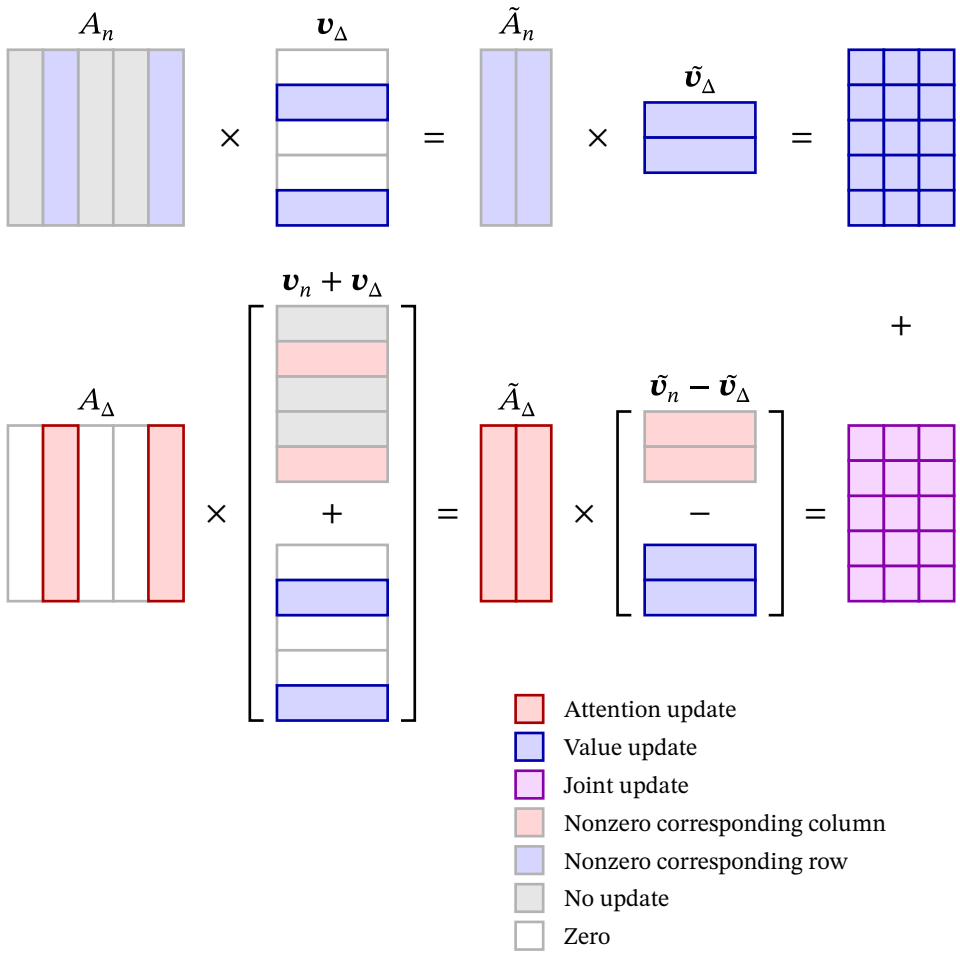


Figure 4.6: **The attention-value product.** We propose a delta-based strategy for sparsely updating the product $A\boldsymbol{v}$. We reduce the cost of each sub-product by cutting rows and columns that do not contribute to the result (due to a zero multiplication).

to produce $\tilde{\boldsymbol{v}}_n$. We then compute

$$\tilde{A}_n \tilde{\boldsymbol{v}}_\Delta + \tilde{A}_\Delta \tilde{\boldsymbol{v}}_n - \tilde{A}_\Delta \tilde{\boldsymbol{v}}_\Delta, \tag{4.9}$$

adding the result to the previous value of $A\boldsymbol{v}$.

The product $\tilde{A}_\Delta \tilde{\boldsymbol{v}}_\Delta$ assumes the columns of $\tilde{A}_\Delta$ are correctly aligned with the rows of $\tilde{\boldsymbol{v}}_\Delta$. We achieve this alignment by forcing the A gate to select the same indices as the $\boldsymbol{v}$ gate. Using a separate policy in the A gate would be possible, but would require a re-alignment operation before computing $\tilde{A}_\Delta \tilde{\boldsymbol{v}}_\Delta$. Further, forcing alignment allows us to eliminate a multiplication by rearranging Equation 4.9 as

$$\tilde{A}_n \tilde{\boldsymbol{v}}_\Delta + \tilde{A}_\Delta (\tilde{\boldsymbol{v}}_n - \tilde{\boldsymbol{v}}_\Delta). \tag{4.10}$$

Equation 4.10 has a cost of $2MND$ (assuming the addition has a negligible cost), compared to N^2D for a standard multiplication. We obtain computation savings when $M < N/2$.

4.4.3 Token Selection Policies

An important design choice for an Eventful Transformer is the token selection policy. Given a gate error tensor $\boldsymbol{e}$, a policy generates a mask $\boldsymbol{m}$ indicating which tokens should be updated. We now discuss policy design.

Top-r policy. This policy selects the r tokens whose error $\boldsymbol{e}$ has the largest norm (we use the L2 norm). The top-r policy is lightweight and has a single parameter that can be easily tuned by hand. Varying r gives direct control over the model's computation cost. These properties make the top-r policy a good fit for applications with tight (potentially time-varying) computational constraints. We use a top-r policy in our main experiments.

Threshold policy. This policy selects all tokens where the norm of the error

$\boldsymbol{e}$ exceeds a threshold h. A threshold policy is input-adaptive; the number of tokens selected depends on the amount of change in the scene. This input adaptivity can potentially lead to a better accuracy-cost tradeoff. However, the best value for the threshold h depends on the distribution of token vectors (which varies across layers) and is difficult to decide. In addition, a threshold policy does not give a fixed compute cost. This policy is likely better suited to applications with more flexible resources, where achieving the best possible accuracy-cost tradeoff is critical.

Other policies. More sophisticated token selection policies could lead to an improved accuracy-cost tradeoff. For example, we could use a learned policy (e.g., a lightweight policy network). However, training the policy's decision mechanism might be challenging, due to the general non-differentiability of the binary mask $\boldsymbol{m}$. Another idea is to use an importance score (e.g., as proposed in [56]) to inform the selection. We leave these ideas as potential topics for future work.

4.5 Experiments

In this section, we present our experiments and results. We evaluate our method for video object detection (Section 4.5.1) and video action recognition (Section 4.5.2). We show additional analysis in Section 4.5.3.

4.5.1 Video Object Detection

Task and dataset. We test our method on video object detection using the ILSVRC 2015 ImageNet VID dataset [197]. We report results on the validation set, which contains 555 videos with lengths of up to 2895 frames. Following prior works [38, 190], we evaluate the mean average precision (mAP) metric with an IoU threshold of 0.5.

Implementation details. We consider the ViTDet model from [131], which we apply to individual frames of an input video. ViTDet combines a Transformer backbone (based on ViT-B [48]) with a standard detection head [24, 79]. The backbone consists of 12 blocks with interleaved global and windowed self-attention (blocks 3, 6, 9, and 12 use global attention). Windowed self-attention uses a window size of 14×14 tokens (224×224 pixels). Token vectors are 768-dimensional. Self-attention operators have 12 heads and employ learned relative position embeddings.

Before the backbone, the model maps each 16×16 image patch to a token vector using a linear transform. The model expects fixed-size inputs (due to resolution-specific position embeddings). Therefore, following from [131], we rescale and pad all video frames to a uniform size (e.g., 1024×1024) before applying the model.

We convert the model to an Eventful Transformer following the method in Section 4.4. In blocks that use windowed attention, we exploit temporal redundancy only within token-wise operations (not within the query-key or attention-value products). Our complete approach is compatible with windowed attention; however, windowing leads to some implementation challenges (ragged tensor shapes across windows, making batched computation more difficult). Note that for ViTDet, global self-attention represents the bulk of the self-attention compute cost.

Experiment protocol and baselines. We fine-tune the original ViTDet weights (trained on COCO) for VID object detection. See Section C.3 for training parameters. Note that we fine-tune *before* we add temporal redundancy awareness to the model. We train and evaluate at resolution 1024×1024. To understand the effect of token count (which strongly influences compute cost), we also evaluate at resolution 672×672. Rather than training a separate lower-resolution model, we adapt the 1024×1024 model by interpolating the position embeddings. The resulting model retains most of its accuracy.

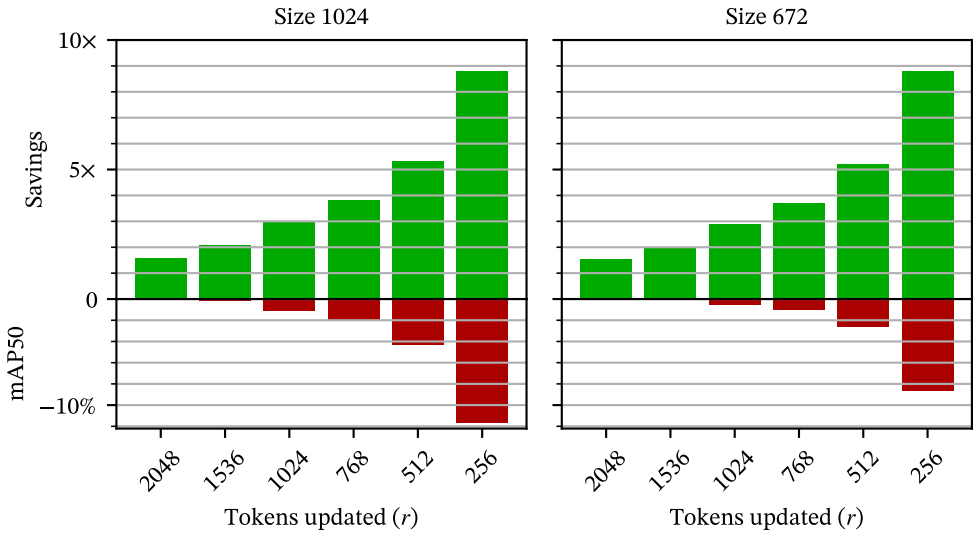


Figure 4.7: **Video object detection results.** Computation savings ratio (positive axis) and relative reductions in mAP50 score (negative axis) for our method. Results are for the ViTDet model [131] on the VID [197] dataset. See the supplement to our paper [51] for tables.

We compare against a version of the STGT method [132]. Due to unavailable source code, we were unable to evaluate all components of this method (notably, the use of a learned policy network). Instead, we consider a simplified version that uses the same top-r policy as our method. This setup enables a direct comparison of the core gating and update mechanisms. In addition, we evaluate an ablated version of our approach that only accelerates token-wise operations. We vary the policy r to explore the accuracy-compute tradeoff. At resolution 1024, we test $r = 256$, 512, 768, 1024, 1536, and 2048 (from a maximum of 4096 tokens). At resolution 672, we test $r = 128$, 256, 384, 512, 768, and 1024 (from a maximum of 1764).

Results. Figure 4.7 shows our results. Our method gives significant savings with only minor reductions in accuracy. For example, at size 1024 with $r = 768$, our approach reduces the cost from 467.4 GFlops to 122.3 GFlops (3.8x lower)

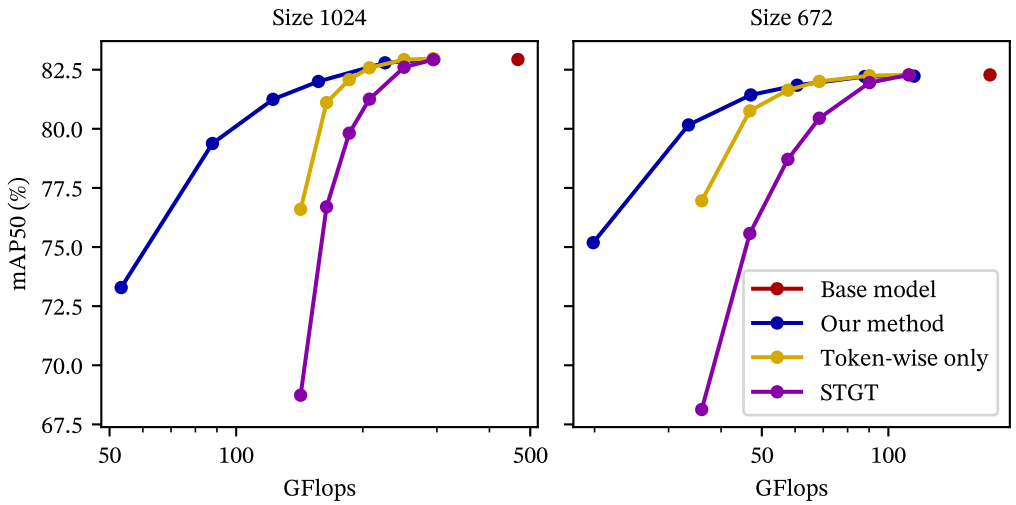


Figure 4.8: **Video object detection comparison and ablation.** The accuracy-cost tradeoff for our method, compared with STGT [132] and an ablation that only accelerates token-wise operations. See the supplement to our paper [51] for tables.

while reducing the mAP50 score from 82.93 to 81.25 (-1.68% in absolute mAP50). At size 672 with $r = 384$, we reduce the cost by 3.7x with a -0.85% change in mAP50.

In these experiments, some tokens correspond to padded space and are therefore "easy" from a temporal redundancy standpoint. However, even in a padding-free deployment (e.g., with a single, known training and inference resolution) our method would still give strong computation savings. For example, consider resolution 1024 with $r = 768$. We are skipping 66% of all non-padded tokens here (based on a measured mean padding ratio of 44.6% on VID—corresponding to a ∼16:9 aspect ratio). This corresponds to a savings of >2x, with an accuracy drop of only 1.68%. Note that our ViViT experiments (Sections 4.5.2 and C.2) do not involve padding.

Figure 4.8 shows the accuracy-compute tradeoff for our method, along with baselines. Our approach gives a considerable improvement in the accuracy-

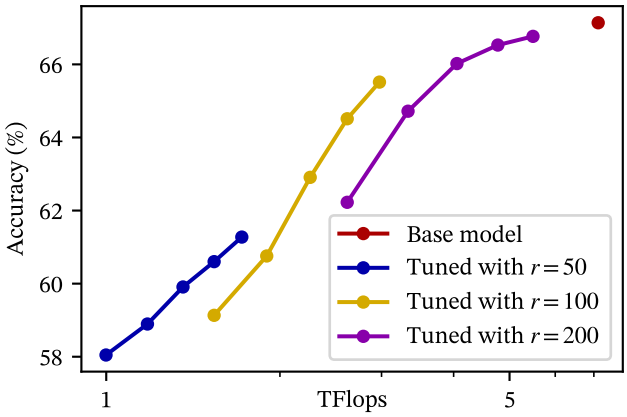


Figure 4.9: **Video action recognition results.** Our results for action recognition on EPIC-Kitchens 100 using the ViViT model [8]. We report the total TFlops per video (spatial + temporal sub-models). See the supplement to our paper [51] for a table containing this data.

compute tradeoff compared to STGT [132]. Further, adding redundancy awareness to the query-key and attention-value products reduces the cost significantly, especially at low r values.

4.5.2 Video Action Recognition

Task and dataset. We evaluate our method on action recognition using the EPIC-Kitchens 100 dataset [45]. EPIC-Kitchens 100 contains highly dynamic egocentric videos annotated with 97 verb and 300 noun classes. We consider the verb classification task. The training and validation set contains 67217 and 9668 action instances, respectively.

Implementation details. We use the ViViT model [8] with factorized spatial and temporal sub-models based on ViT-B. The spatial sub-model (the bulk of the compute cost) is applied sequentially to 16 2-frame input clips. The outputs of the spatial model are concatenated and passed to the temporal model, which returns a class prediction. The prediction is the average over 12

video views (4 temporal views, each divided into 3 spatial crops). Each view has a shape of 320×320×32. Unlike ViTDet, ViViT adds a class embedding token (see [48]), does not use windowed self-attention, and does not use relative position embeddings.

We convert the spatial model to an Eventful Transformer. Naively replacing the spatial model with an Eventful version leads to a considerable drop in accuracy (about -10% with $r = 100$). We conjecture that the cause is a distribution shift in the inputs to the temporal model (see Section C.1 for further discussion). We recover most of the lost accuracy by fine-tuning the *non-Eventful* temporal model on the outputs of a frozen Eventful spatial model.

Experiment protocol. We start with ViViT pre-trained on EPIC-Kitchens 100 and fine-tune the temporal model as described above (on the EPIC-Kitchens training set). We fine-tune different model variants with policy r values of 50, 100, and 200 (out of a maximum of 401 tokens). See Section C.3 for training parameters. We report results using the top-1 accuracy metric, following standard protocol [45].

Results. Figure 4.9 shows our results for the Eventful ViViT model. We evaluate a range of r values for each of the fine-tuned variants. We test the original fine-tuned r-value, along with ±20% and ±40% of this value. We observe considerable computation savings with only moderate reductions in accuracy. For example, with $r = 140$, we reduce the cost by 2.4× while reducing the accuracy by only 1.62%. In addition, the model retains adaptivity despite being fine-tuned with a single-r value, exhibiting a favorable accuracy-compute tradeoff over a range of r-values.

4.5.3 Spatial Redundancy and Runtime

Considering spatial redundancy. Eventful Transformers exploit *temporal* redundancy and thus complement prior works that leverage *spatial* redun-

Table 4.1: **Adding spatial redundancy to ViTDet.** "Spatial" is a model with pooling in $\boldsymbol{k}$ and $\boldsymbol{v}$. "Spatiotemporal" is a model with both pooling and temporal redundancy awareness.

Variant	r	mAP50 (%)	GFlops
Base model	—	82.93	467.4
Spatial	—	80.15	388.1
Spatiotemporal	2048	80.14	217.0
Spatiotemporal	1536	80.07	169.3
Spatiotemporal	1024	79.50	121.0
Spatiotemporal	768	78.69	96.3
Spatiotemporal	512	76.96	70.9
Spatiotemporal	256	71.35	44.5

dancy. Here we present a simple proof-of-concept experiment that considers spatial redundancy in Eventful Transformers.

Specifically, we adopt a variant of [251], which applies spatial pooling to the self-attention key and value tokens. We apply this method with 2×2 pooling to the global self-attention operators in the ViTDet model. We evaluate this method both with and without temporal redundancy awareness. In the temporal-redundancy model, we pool $\boldsymbol{k}$ and $\boldsymbol{v}$ after their respective buffers. We pool $\tilde{\boldsymbol{k}}$ by first pooling the active indices (equivalent to max-pooling the mask $\boldsymbol{m}$), then gathering the buffered $\boldsymbol{k}$ using the pooled indices.

Table 4.1 shows our results for resolution 1024 (See the supplement to our paper [51] for resolution 672). We see that the spatial and temporal methods are complementary; both meaningfully contribute to reducing the computational cost. See Section 4.6 for further discussion on spatial redundancy methods.

Runtime. We show preliminary runtime results on a CPU (Xeon Silver 4214, 2.2 GHz) and a GPU (NVIDIA RTX 3090). See Section C.3 for experiment details. Table 4.2 shows our results. Adding temporal redundancy awareness leads to speedups of up to 1.74× on the GPU and 2.48× on the CPU. These results should be seen just as a proof of concept—we are confident that these speedups could be improved with further engineering effort (e.g., by replacing

Table 4.2: **Runtimes (ms).** ViTDet runtimes are for the Transformer backbone only. ViViT runtimes include the temporal sub-model.

Model	Size	Variant	r	GPU	CPU
ViTDet	1024	Base model	—	86.6	5150
ViTDet	1024	Spatial	—	58.9	3116
ViTDet	1024	Temporal	512	69.9	3570
ViTDet	1024	Spatiotemporal	512	38.1	1682
ViTDet	672	Base model	—	28.3	1492
ViTDet	672	Spatial	—	23.3	1055
ViTDet	672	Temporal	256	21.6	838
ViTDet	672	Spatiotemporal	256	20.8	478
ViViT	320	Base model	—	950	5.45×10^4
ViViT	320	Temporal	50	545	2.15×10^4

vanilla PyTorch operators with custom kernels or using a high-performance inference framework).

Update visualization. Figure 4.10 shows an example video sequence. We visualize the model predictions (top), the token-wise L2 norm of the error $\boldsymbol{e}$ (middle), and the update mask $\boldsymbol{m}$ (bottom). We see that larger error values correspond to dynamic regions in the image.

4.6 Discussion

Memory overhead. Our method reduces floating point operations at the cost of higher memory usage. Each gate or buffer maintains a reference tensor ($\boldsymbol{u}$ or $\boldsymbol{b}$, respectively). The memory overhead for token gates and buffers is generally modest. For, consider size-1024 ViTDet. The model has 4096 768-dimensional tokens, meaning a token gate or buffer takes 12.6/6.3 MB of memory at full/half precision.

However, gating or buffering the attention matrix A can require a larger amount of memory. For example, in the global attention layers of the size-1024 ViTDet model, the A matrix has shape 4096×4096×12. Buffering this A requires 805/403 MB at full/half precision. Fortunately, the situation dramati-

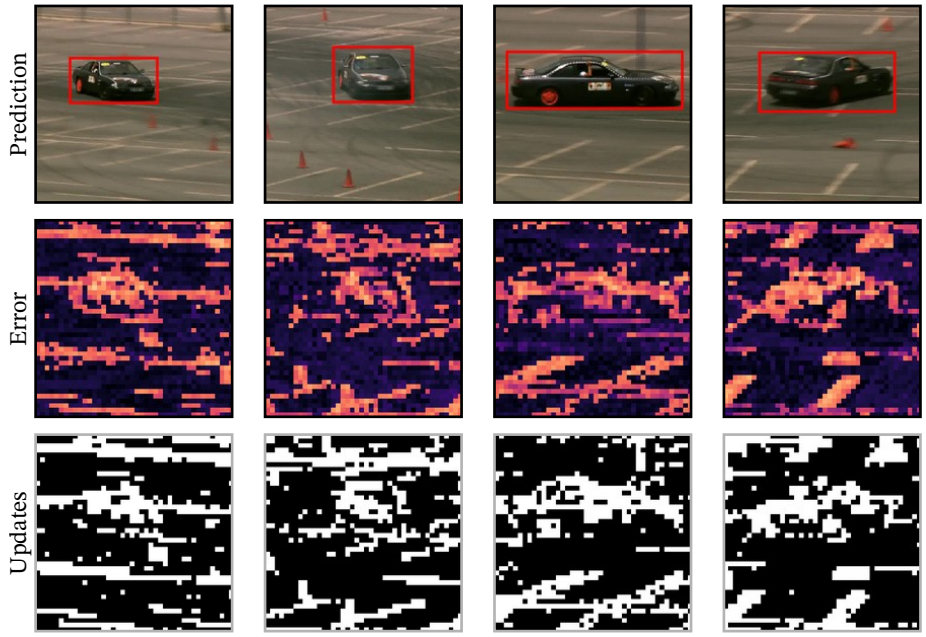


Figure 4.10: **Update visualization.** The error $\boldsymbol{e}$ and update mask $\boldsymbol{m}$, for the pre-QKV gate in block 3 of ViTDet. Video source: [197]

cally improves if we reduce the number of tokens or use windowed attention (due to the quadratic size of A). For example, the A matrices for the size-672 ViTDet model (1764 tokens) and the ViViT model (301 tokens) occupy 149/75 MB and 4.3/2.2 MB, respectively. In applications where memory is tight and the A matrix is large, it is possible to save memory by removing temporal redundancy awareness in the query-key and/or attention-value products (each eliminates one A-shaped state tensor).

Integration with spatial redundancy methods. A promising avenue for future work the is further integration of our approach with spatial redundancy methods. Conceptually, these methods summarize a large, redundant set of tokens using a more compact set of tokens. The gating module in an Eventful Transformer assumes that most of its inputs vary smoothly over time. When combining our approach with spatial redundancy methods, we need to ensure

that the compact spatial summary is relatively stable. For example, with adaptive token clustering [17, 156], we would need to sort the clusters in a mostly-consistent order.

There are many potentially interesting questions regarding joint modeling of spatial and temporal redundancy. For example, how is the temporal axis different from the spatial one? Should the temporal axis be modeled separately? We leave such questions for future work.

5 Stability and Robustness: Instant Video Models

5.1 Introduction

Video is often processed *frame-wise*—meaning images are passed one by one to a processing pipeline or model, and each output is independent of the previous one. This design choice is often driven by practical considerations: single-frame datasets are generally more diverse and accessible than video datasets, training image-based models is far less demanding in terms of compute and memory, and improvements in single-frame performance often carry over to video-based tasks.

Unfortunately, frame-wise processing faces the inherent challenge of *temporal consistency*, where model predictions fluctuate over time (Figure 5.1 (top-middle)). This behavior is especially problematic in tasks such as denoising or stylization, where temporal inconsistency can significantly reduce perceptual quality. Even in applications where the model output is not intended for human consumption, instability can impact downstream tasks. For example, inconsistent monocular depth estimates could produce erratic behavior in a collision avoidance system. Moreover, instability can reduce perceived reliability and thereby undermine user trust, regardless of objective performance.

Temporal consistency is closely related to *corruption robustness*. In field de-

ployments, vision systems often operate in non-ideal conditions. For example, an autonomous vehicle may encounter inclement weather, sensor artifacts, or low-light noise; robustness in these circumstances is vital for safe system operation. These real-world corruptions are often *transient*, with an appearance that changes between frames (Figure 5.1 (top-right)). While it may be challenging to correct such degradations with a single frame, we can often infer the underlying clean signal from recent context. In this case, we can view robustness as a natural extension of temporal consistency.

Several prior works have proposed video-centric models with improved temporal consistency [18, 144, 89, 255, 210, 250]. However, these methods are often narrowly designed for one or a few tasks and require costly training on large-scale video datasets. Consequently, they lack the flexibility to leverage the extensive ecosystem of frame-based imaging and perception models. Further, few explicitly address robustness to transient corruptions or other challenging conditions.

In this work, we improve the temporal consistency and robustness of *pre-trained, image-based* models across various tasks. One of our primary challenges is that increasing stability may result in over-smoothing, which can, in turn, reduce accuracy. We conceptually explore the tradeoffs between output quality, corruption robustness, and temporal consistency, and introduce a unified accuracy-robustness-stability loss to balance these objectives. We provide a theoretical analysis of this loss and identify strategies to avoid "over-smoothing reality," such that there is no incentive for predictions to be smoother than the true scene dynamics.

Guided by this analysis, we propose a class of versatile *stabilization adapters* (Figure 5.1 (middle)). These adapters generate control signals, based on recent spatiotemporal context, that modulate changes to the model's features and output. By operating in both the feature and output spaces, we allow the adapters to model stability wherever it exists in the visual hierarchy. This

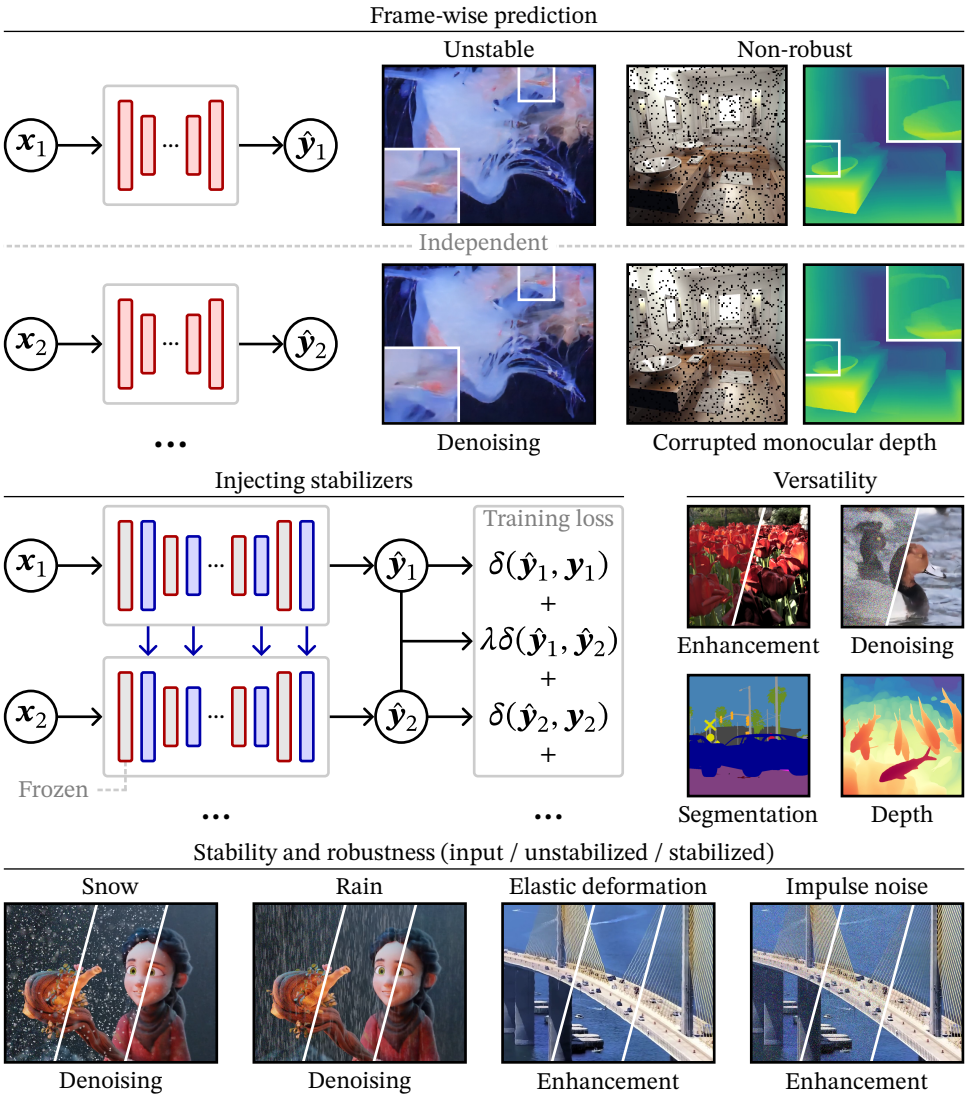


Figure 5.1: **Stabilizing image-based networks. (top)** Single-image models can give unstable predictions and failures under time-varying corruptions. In the top-right example, we see that randomly dropping patches causes artifacts in monocular depth estimates. **(middle)** We propose a method for injecting stabilizers into existing networks and for training these stabilizers using a unified accuracy-stability-robustness loss. **(bottom)** We demonstrate improvements in stability and robustness for various tasks, without modifying the original image-based models. Image sources: [64, 192, 203, 258].

property is important for high-level vision tasks, where stability is often best described in a feature space.

Our method offers several key benefits. First, our stabilization adapters are lightweight and modular, and do not require modifying the original model parameters. Second, our adapters operate causally; stabilized outputs depend only on current and past inputs—a feature that is critical for processing streaming video in latency-sensitive applications. Third, our approach is compatible with both low-level tasks, where stability can be described in terms of pixel values, and higher-level tasks, where stability occurs at the level of scene semantics. Finally, our method naturally enhances robustness to transient corruptions, without requiring explicit corruption modeling.

We evaluate our method on a range of tasks: denoising, image enhancement, monocular depth estimation, and semantic segmentation (Figure 5.1 (middle-right)). We also demonstrate improved robustness against various transient corruptions, including noise, dropped patches, elastic deformations, compression artifacts, and adverse weather (Figure 5.1 (bottom)). In most cases, these improvements do not reduce accuracy—on the contrary, we often see significant improvements in task metrics. Overall, our experiments establish the flexibility and practicality of our approach.

5.2 Related Work

Corruption robustness. Several prior works have addressed robustness against input corruptions. Hendrycks and Dietterich [81] propose metrics for measuring the robustness of image classifiers against common corruptions (e.g., compression artifacts or weather); their metrics inspire our definitions in Section 5.3. In general, natural corruptions have received less attention [49] from the vision community than adversarial corruptions [2, 66, 154, 174, 205, 219, 238, 249], although a handful of methods and benchmarks exist [12, 119,

140, 163, 244]. Like these works, our paper emphasizes robustness to naturally occurring corruptions rather than worst-case adversarial perturbations.

Consistent image enhancement. Frame-to-frame flickering is a significant problem for low-level image enhancement models; as such, there have been several works that improve temporal consistency for these tasks [18, 118, 125, 261, 268]. Blind video temporal consistency methods [18, 118, 125] treat the frame-level model as a black box, which allows generalization across models and applications. However, because they consider only the model input and output, these methods cannot model higher-level (semantic) stability and are prone to instability when the input is impacted by transient noise or corruptions. In contrast, we model stability in both the output space and the feature space, improving robustness against corrupted inputs and supporting a wider range of tasks, including those where stability is best described in semantic terms.

Task-specific video architectures. Many works propose video-optimized architectures for specific tasks, with temporal consistency a stated priority in many cases. This problem has been widely studied for ill-posed, low-level tasks where the output is intended for a human viewer; examples include video colorization [124, 144, 228, 260, 267, 272, 273], stylization [33, 36, 37, 47, 59, 60, 71, 89, 130, 142, 183, 196, 235, 252], and inpainting [32, 111, 123, 229, 255, 266]. Temporal consistency is also a concern in higher-level tasks, including segmentation [9, 180, 188, 210, 230], object detection [14, 102, 141, 223, 250, 275], and depth estimation [104, 108, 117, 129, 150, 179, 237, 269]. Clockwork ConvNets [210] leverage the observation that semantic content evolves more slowly and smoothly than pixel values; we share this motivation in designing feature-domain stabilizers.

Our goal is not to design a task-specific method; in fact, we expect specialized architectures to outperform our general approach on benchmarks. The appeal of our approach lies in its practicality and versatility. Our method requires

minimal training and no alterations to the original network, and can be applied to a broad range of video inference tasks, including those where highly optimized video architectures may not exist.

5.3 Defining Stability and Robustness

We start by defining temporal stability. Let $f_\phi : \mathcal{X} \to \mathcal{Y}$ be a frame-wise predictor with parameters ϕ, and let $\delta(\boldsymbol{y}_1, \boldsymbol{y}_2)$ be a metric defined on the space $\mathcal{Y}$. We use $\hat{\boldsymbol{y}}$ to indicate the model output and $\boldsymbol{y}$ for the target output (ground truth, or features from a reference model). We formulate our definition as an expectation over data distribution $\mathcal{D}$ containing $(\boldsymbol{x}, \boldsymbol{y})$ sequences of duration τ indexed by discrete time step t (the frame index). We define the stability $\mathcal{S}$ as the negative expected difference between adjacent predictions, i.e.,

$$\mathcal{S} = -\mathbb{E}_{(\boldsymbol{x},\boldsymbol{y},\tau)\sim\mathcal{D}}\left[\sum_{t=1}^{\tau-1} \delta(f_\phi(\boldsymbol{x}_t), f_\phi(\boldsymbol{x}_{t+1}))\right]. \tag{5.1}$$

The negation is added such that stability increases as change decreases.

This notion of stability is closely related to robustness, i.e., the correctness of the model predictions under input corruptions [81]. The same input corruptions can cause both temporal instability and reduced prediction accuracy; examples include sensor noise, image or video compression artifacts, rain, and snow. Hendrycks and Dietterich [81] define corruption robustness $\mathcal{R}_c$ as the expected accuracy of a classifier under a distribution $\mathcal{E}$ of per-image perturbation functions. We extend their definition to cover arbitrary metrics δ and time series of duration τ,

$$\mathcal{R}_c = -\mathbb{E}_{\varepsilon\sim\mathcal{E},(\boldsymbol{x},\boldsymbol{y},\tau)\sim\mathcal{D}}\left[\sum_{t=1}^{\tau} \delta\left(f_\phi(\varepsilon_t(\boldsymbol{x}_t)), \boldsymbol{y}_t\right)\right], \tag{5.2}$$

where ε_t is the per-frame perturbation at time t. Again, $\boldsymbol{y}_t$ is the target output.

We define the corruption stability $\mathcal{S}_c$ similarly:

$$\mathcal{S}_c = -\mathbb{E}_{\varepsilon\sim\mathcal{E},(\boldsymbol{x},\boldsymbol{y},\tau)\sim\mathcal{D}}\left[\sum_{t=1}^{\tau-1}\delta(f_\phi(\varepsilon_t(\boldsymbol{x}_t)), f_\phi(\varepsilon_{t+1}(\boldsymbol{x}_{t+1})))\right]. \tag{5.3}$$

Both $\mathcal{R}_c$ and $\mathcal{S}_c$ include input perturbations ε. $\mathcal{R}_c$ measures how accurately a model f predicts the target, while $\mathcal{S}_c$ captures the temporal smoothness of the model's outputs. Notably, $\mathcal{R}_c$ and $\mathcal{S}_c$ can be applied to both the intermediate features and the output of f.

5.4 Learning to Balance Stability and Robustness

We now combine $\mathcal{R}_c$ and $\mathcal{S}_c$ to form a unified accuracy-stability-robustness training loss $\mathcal{U}_c$, and analyze the conditions under which this loss leads to well-behaved training.

Unified accuracy-stability-robustness loss. We define the unified loss as

$$\mathcal{U}_c = -(\mathcal{R}_c + \lambda\mathcal{S}_c) \tag{5.4}$$

$$= \mathbb{E}_{\varepsilon\sim\mathcal{E},(\boldsymbol{x},\boldsymbol{y},\tau)\sim\mathcal{D}}\left[\sum_{t=1}^{\tau}\delta(f_\phi(\varepsilon_t(\boldsymbol{x}_t)), \boldsymbol{y}_t) + \lambda\sum_{t=1}^{\tau-1}\delta(f_\phi(\varepsilon_t(\boldsymbol{x}_t)), f_\phi(\varepsilon_{t+1}(\boldsymbol{x}_{t+1})))\right], \tag{5.5}$$

where λ is a constant that weights stability relative to accuracy.

Theoretical analysis. If δ can be expressed in terms of a norm on $\mathcal{Y}$, we can derive two bounds on λ. The first, $\lambda < 1/2$, which we call the *oracle bound*, defines the range of λ where the ground truth is the global minimizer of the loss in prediction space. Under this bound, a perfectly accurate (oracle) model will never have an incentive to diverge from the correct prediction to increase stability. For each training item $\boldsymbol{x}$, the minimum loss occurs at the

ground truth $\boldsymbol{y}$, implying zero gradients with respect to the prediction $\hat{\boldsymbol{y}}$. See Appendix D.1 for a proof of this and the following bound.

The second bound, $\lambda > \tau - 1$, is the *collapse bound*, and gives the range of λ where the global loss minimizer corresponds to exact repetition of the initial prediction, regardless of scene changes. Unlike the global accuracy minimizer (the oracle state), which may be difficult or impossible to reach with gradient descent, the collapse state is often easily achieved. For example, if there is an EMA stabilizer (Section 5.5) on the output, we can achieve collapse simply by setting its decay to zero. In our experiments, we confirm that setting λ above the collapse bound leads to prediction collapse, provided the collapse state is representable in the stabilizer parameter space.

Example. In Figure 5.2, we plot the loss as a function of two one-dimensional predictions $\hat{y}_2$ and $\hat{y}_3$, for several values of λ. When the stability penalty is introduced, the loss begins to tilt toward more stable predictions. When λ is within the oracle bound, the global minimizer is unchanged. When it exceeds the collapse bound, the global minimizer is a repeated prediction.

Note that the oracle state is mutually exclusive with the collapse state (unless the ground truth is itself collapsed), as for any time series with $\tau > 1$, we have $\tau - 1 > 0.5$. Thus, we recommend training with $\lambda < 0.5$ in general; doing so ensures non-collapse and yields the correct behavior in the limiting case where the model is perfectly accurate.

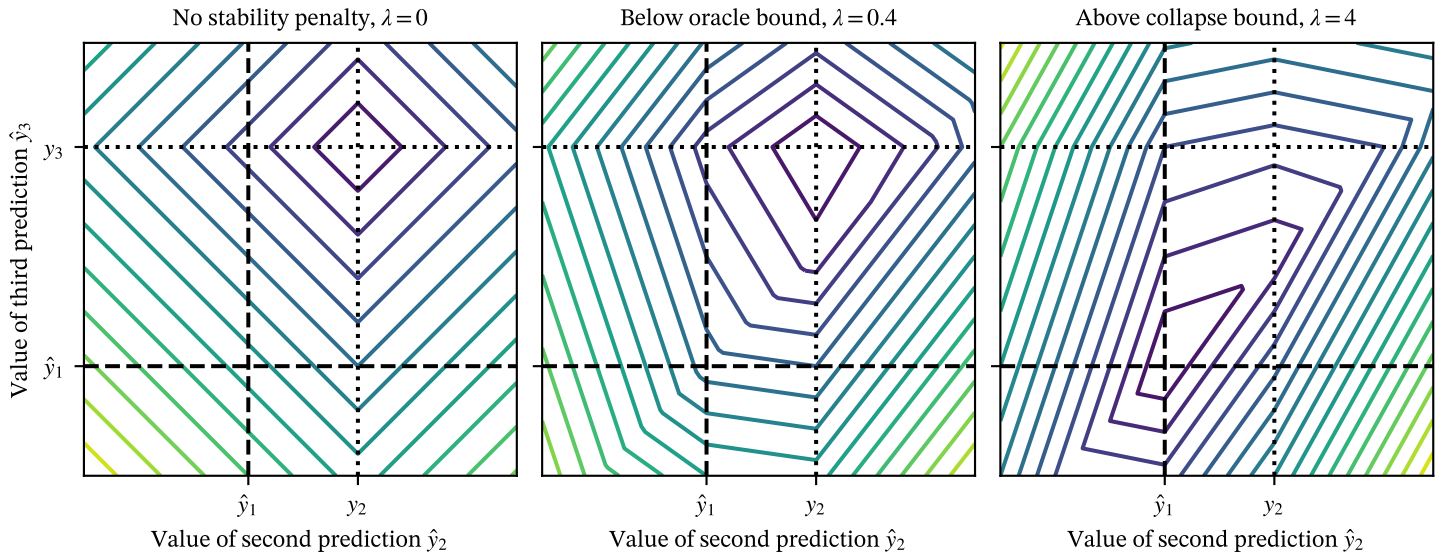


Figure 5.2: **Unified loss for one-dimensional predictions.** We consider a time series of duration $\tau = 3$ consisting of one-dimensional predictions, with δ defined as the L1 distance. We assume that the first prediction $\hat{y}_1$ is fixed (cannot be modified by a stabilization adapter). We show the value of the second prediction $\hat{y}_2$ along the x-axis and the value of the third $\hat{y}_3$ along the y-axis, with contours indicating the value of the unified loss as these predictions vary. When $\lambda = 0$, the minimum occurs at the ground truth $\hat{y}_2 = y_2$ and $\hat{y}_3 = y_3$. When λ is nonzero but below the oracle bound, the minimum still occurs at the ground truth, but the loss increases more slowly in the direction of stabler predictions. When λ exceeds the collapse bound, the global minimum is the collapse state $\hat{y}_3 = \hat{y}_2 = y_1$.

5.5 Designing Stabilization Adapters

Our goal is to improve the temporal stability and robustness of a pre-trained, frame-wise predictor f for video tasks. We assume f is realized using a deep neural network with pre-trained weights ϕ_0. While the unified loss $\mathcal{U}_c$ allows us to update ϕ_0, this fine-tuning may be computationally expensive or require substantial training data. Instead, we consider adaptation of f, where a task-specific, lightweight adapter parametrized by $\Delta\phi$ is learned to stabilize the intermediate features and outputs of f. Under this formulation, we train only the parameters $\Delta\phi$ of the adapter (i.e., the stabilizer), and the original weights ϕ_0 remain fixed.

Design principles. The following principles guide our stabilizer design. *First*, we consider only causal stabilizers, where the stabilized outputs at time t are computed exclusively using information from times $\leq t$. This constraint is critical for processing streaming videos. *Second*, we stabilize both network activations (features) and outputs. Output-only stabilization is sufficient for some low-level operations such as colorization. Feature-domain stabilization expands the potential scope of our method to include higher-level tasks; for these tasks, the inherent stability of a scene is often best described in feature space. *Finally*, we limit ourselves to designs that do not interfere with the existing network architecture. Our stabilizers are layer-level adapters with independent parameters, designed to preserve the existing representation.

Exponential moving average (EMA) stabilizer. As a starting point, we consider a simple temporal smoothing operation applied to individual feature values forming a one-dimensional time series. Let z_t denote an activation or feature value at time t, and let $\tilde{z}_t$ denote the corresponding stabilized activation. The exponential moving average (EMA) stabilizer produces a linear combination of the current unstabilized and previous stabilized outputs,

$$\tilde{z}_t = \beta z_t + (1-\beta)\tilde{z}_{t-1}, \tag{5.6}$$

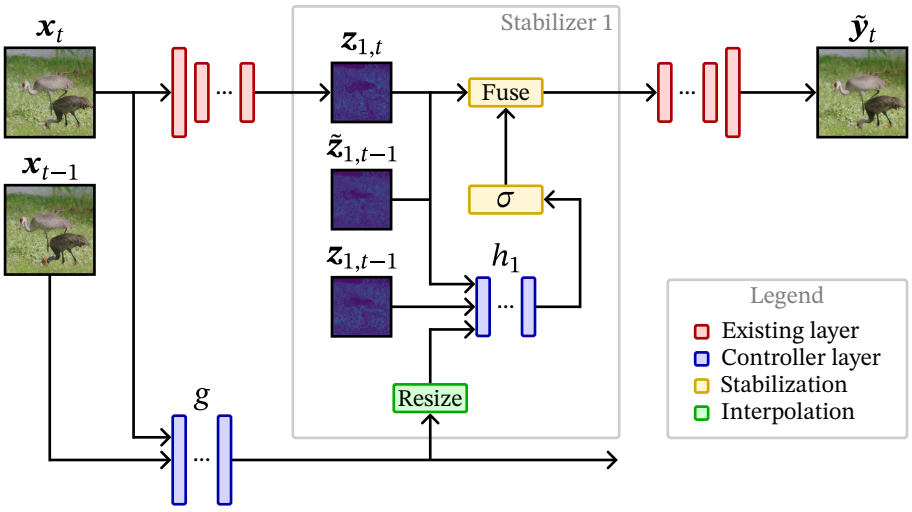


Figure 5.3: **Stabilization controllers.** Starting with the existing network (red), we add stabilizers (yellow) to select layers. The degree of stabilization, i.e., the decay β, can be predicted by a stabilization controller (blue). This controller consists of a shared backbone g and one head h_i per stabilized layer. Stabilizers can be added to both internal layers and the model output.

where $\beta \in [0, 1]$ is a decay-rate parameter. The recursive formulation here is equivalent to convolving the input time series with an infinite exponentially decaying weight kernel. The EMA stabilizer is memory-efficient (only $\tilde{z}_{t-1}$ is retained) and differentiable with respect to β and z. We use the EMA stabilizer as the basis for more sophisticated designs.

Stabilization controllers. Simple smoothing operations (like the EMA stabilizer) are limited in both their spatial context and their ability to model complex changes in the scene. To address these limitations, we propose a *stabilization controller network*, which considers prior context (e.g., the current and previous input frames and feature maps) and predicts the amount of stabilization that should be applied to each value.

Although the idea of controller-augmented stabilization can be applied to many stabilization mechanisms, we focus here on the EMA stabilizer due to

its differentiability and low memory requirements. In this case, the controller predicts the decay β for each activation across layers. Our architecture, as shown in Figure 5.3, consists of a shared backbone g and a stabilization head h_i per stabilized layer. Tog g compares the current and previous frames, offering a shortcut connection from frames to features. h predicts the decay values based on the output of g (resized to match the layer resolution), the current unstabilized features z_t, and the previous stabilized and unstabilized features $\tilde{z}_{t-1}$ and z_{t-1}. Together, the parameters of g and $\{h_i\}$ form $\Delta\phi$.

Formally, the stabilized feature tensor $\tilde{\boldsymbol{z}}_{i,t}$ for layer i and time t is given by

$$\tilde{\boldsymbol{z}}_{i,t} = \boldsymbol{\beta}_{i,t} \odot \boldsymbol{z}_{i,t} + (1 - \boldsymbol{\beta}_{i,t}) \odot \tilde{\boldsymbol{z}}_{i,t-1}, \tag{5.7}$$

$$\boldsymbol{\beta}_{i,t} = \sigma(h_i(g(\boldsymbol{x}_t, \boldsymbol{x}_{t-1}), \boldsymbol{z}_{i,t}, \tilde{\boldsymbol{z}}_{i,t-1}, \boldsymbol{z}_{i,t-1})), \tag{5.8}$$

where $\odot$ is an element-wise product and $\boldsymbol{z}_{i,t}$ is the unstabilized feature tensor.

We note that our controller is conceptually similar to selective state space models [69], although the design differs significantly. Equation 5.7 can be viewed as a linear dynamical system defined on frame-level features with parameters conditioned on the input, current, and previous features.

Controller with spatial fusion. Often, $\boldsymbol{z}$ takes the form of a 2D feature map. If g and h are convolutional, the predicted decay β is informed by the frames and features within a local receptive field. However, with Equation 5.8, the weighted fusion used to compute $\tilde{\boldsymbol{z}}$ is still constrained to the time axis. Extending the weighted fusion to a spatial neighborhood can improve stabilization in the presence of motion by allowing feature translation.

To perform spatial fusion, we modify the controller head h to predict a spatial decay kernel $\boldsymbol{\eta}$ at each pixel rather than a single decay β. For a neighborhood that contains m locations (including the central pixel), the kernel $\boldsymbol{\eta}$ contains $m+1$ elements. The first m elements weight the stabilized activations from the previous time step ($\tilde{\boldsymbol{z}}_{t-1}$), for each location in the neighborhood. The $(m+1)$th

element weights the current unstabilized activation ($\boldsymbol{z}_t$) for the central pixel. The kernel is softmax-normalized. The first m logits are predicted directly by the controller head, and the last is set to zero (when $m = 1$, the softmax reduces to the sigmoid in Equation 5.8). See Section D.4.3 for more details.

The spatial fusion stabilizer can represent a recursive shift projection, where a feature vector is translated on each frame by an amount corresponding to the object motion. The maximum trackable motion in this case is determined by the spatial extent of the kernel $\boldsymbol{\eta}$.

5.6 Experiments

In Sections 5.6.1 and 5.6.2, we test our approach on image enhancement and denoising, respectively. We ablate the components of our method and explore the tradeoff between stability and accuracy. In Section 5.6.3, we test our stabilizers in the presence of various image corruptions, evaluating image enhancement, denoising, and depth estimation. In Section 5.6.4, we consider corruptions resulting from adverse weather (rain and snow).

Some low-level details (e.g., training hyperparameters) are omitted here for brevity; see the appendices for a more exhaustive description of experiment protocols. The appendices also include results for semantic segmentation and an exploration of training-free stabilizer composition.

Variants and baselines. Across our experiments, we consider a common set of variations (ablations of our approach) and baselines. The *simple learned* variant adds a simple EMA stabilizer (Equation 5.6) to the output and features, with one learned β value per channel. The *controlled* variant adds a learned controller that predicts the stabilizer decay according to Equation 5.8. Finally, the *spatial* variation augments the controlled stabilizer by adding spatial fusion. As for baselines: the *simple fixed* method applies a simple EMA stabilizer to internal features and the output, with one global, hand-tuned

β; the *output fixed* method applies a hand-tuned EMA stabilizer only to the model output.

5.6.1 Image Enhancement

Task, dataset, and base model. We first consider image enhancement, a task where perceptual quality (including stability) is of primary importance. We use the HDRNet model [64], which can be trained to reproduce many low-level image transformations. Specifically, we target the local Laplacian detail-enhancement operator [177], due to its known forward model and readily available code. We consider two effect strengths: moderate ($\sigma = 0.4$ and $\alpha = 0.5$) and strong ($\sigma = 0.4$ and $\alpha = 0.25$). We generate training pairs by applying the local Laplacian filter to each frame of the Need for Speed (NFS) dataset [110]. NFS contains 100 videos (380k frames) collected at 240 FPS; we randomly select 20 videos for validation and use the remaining 80 for training. Videos are scaled to have a short-edge length of 360. We evaluate PSNR and instability (Equation 5.1) with $\delta = ||\cdot||_2$.

Experiment protocol. We fine-tune the original HDRNet weights (with ID local_laplacian/strong_1024) for both effect strengths. After this fine-tuning, we attach a stabilizer to the output of each convolution and the overall model output (because the HDRNet architecture is extremely lightweight, this does not represent an unreasonable overhead). We then freeze the fine-tuned weights and train the stabilizers using BPTT on short video snippets ($\tau = 8$). We use the unified loss with $\delta = ||\cdot||_2$. We test several λ values—0.1, 0.2, 0.4, 0.8, and 8.0—for each effect strength and model variation. The first three values are within the oracle bound, and the last exceeds the collapse bound.

Results. Figure 5.4 illustrates how PSNR and instability change as we vary the degree of stabilization (λ for learned stabilizers, β for hand-tuned stabilizers). We observe that for static (non-controlled) stabilizers, there is no benefit to stabilizing in feature space. For the static methods (output fixed, simple fixed,

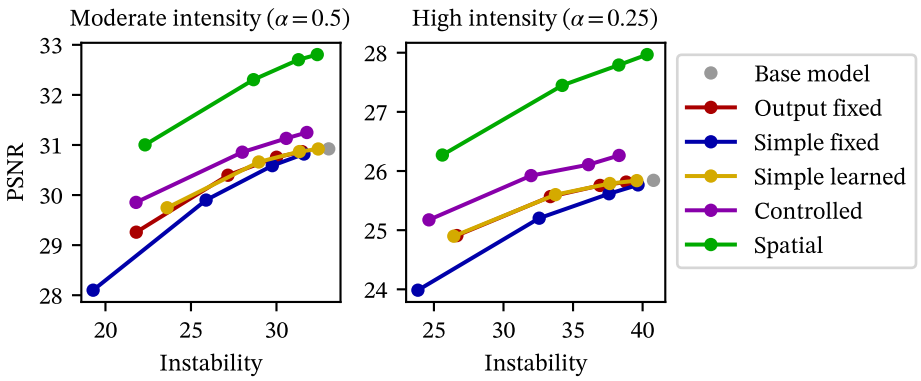


Figure 5.4: **Image enhancement results.** Introducing a controller and spatial fusion to the stabilizer significantly improves the accuracy-stability tradeoff. The spatial-fusion stabilizer reduces frame-to-frame variation by up to $\approx$ 35% while exceeding the quality of the base model. "Instability" here refers to negative stability ($-\mathcal{S}$); see Equation 5.1. The goal is to move toward $-x$ (lower instability) and $+y$ (better image quality).

and simple learned), increasing stability brings a reduction in quality as we begin to over-smooth the output. In contrast, for the controlled and spatial stabilizers, there is a region where both PSNR and stability are improved over the base model. Spatial fusion gives a significant improvement in output quality—roughly 2 dB for the high-intensity effect. We suspect this is related to the nature of the detail-enhancement task (its sensitivity to small-scale motion). Finally, we confirm that setting $\lambda = 8 > \tau - 1$ leads to prediction collapse (instability $< 10^{-3}$, indicating a constant prediction). This result confirms that the global collapse minimum is easily reached, despite the non-convex nature of the optimization in general.

5.6.2 Denoising

Task, dataset, and base model. Next, we evaluate image denoising under AWGN. Denoising highlights the utility of our method when the input is itself unstable. We use the NAFNet model [35], which employs a U-Net

architecture with modified convolutional blocks (NAFBlocks). We again use the NFS dataset, with the same train/validation split as in Section 5.6.1. We evaluate three noise levels: moderate ($\sigma = 0.1$, for float images $\in [0, 1]$), strong ($\sigma = 0.2$), and extreme ($\sigma = 0.6$). See Appendix D.6.4 for additional results on the DAVIS [181] dataset.

Experiment protocol. We start by fine-tuning the unstabilized model for each dataset and noise level, initializing with the nafnet_sidd_width32 weights published by the model authors. We then attach a stabilizer to the output of each NAFBlock and to the model output. As before, we freeze the fine-tuned weights and train only the stabilizer parameters, using the unified loss with $\delta = ||\cdot||_2$ and sweeping out $\lambda = 0.1$, 0.2, 0.4, 0.8, and 8.0.

Results. Figure 5.5 shows PSNR and instability across noise levels. The "simple fixed" stabilizer gives a somewhat surprising result: adding stabilization worsens both PSNR and instability. The reason is that the network backbone predicts a noise residual, which is completely uncorrelated between frames. Over-smoothing this residual inhibits the noise removal, which worsens PSNR and increases frame-to-frame variation due to unremoved noise. Fortunately, this behavior does not exist for the learned or controlled stabilizers, as they target only those features that exhibit some temporal smoothness. Controlled stabilizers improve both stability and PSNR when $\lambda \leq 0.4$. We again confirm that setting $\lambda = 8 > \tau - 1$ leads to prediction collapse, i.e., instability $< 10^{-3}$.

In most cases, we find that the spatial fusion stabilizer outperforms the non-spatial controlled stabilizer. However, there is a notable exception in the case of extreme noise, where the spatial fusion stabilizer is about 6 dB worse than other methods (these points are outside the range of the plot in Figure 5.5 but are included in Table D.5). Intriguingly, this quality gap only appears when evaluating long sequences (hundreds of frames) and shrinks as we reduce τ. In Appendix D.6.6, we show that this problem can be at least partially mitigated by increasing τ during training.

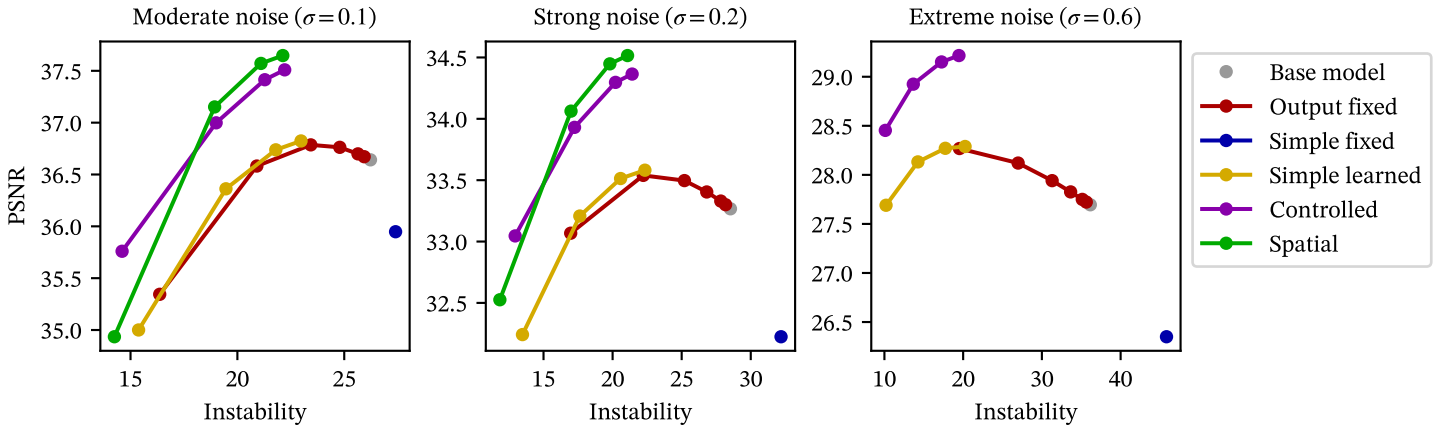


Figure 5.5: **Denoising results.** Because it attempts to stabilize an iid noise residual, naive feature-space stabilization leads to worse PSNR *and* worse stability. We achieve the best performance with a controlled stabilizer, usually with spatial fusion.

The supplementary material includes video files comparing the stabilized and unstabilized outputs. We encourage the reader to watch these videos for a clear qualitative comparison (temporal inconsistency is much more obvious in videos than in side-by-side static frames).

5.6.3 Corruption Robustness

Tasks, datasets, and base models. We again consider HDRNet for image enhancement and NAFNet for denoising. We also include results for depth estimation with Depth Anything v2 [258]. Depth Anything is notable for the scale of data used in its training; it would be costly to develop a new video architecture from scratch, making Depth Anything a good candidate for our approach. See Appendix D.6.2 semantic segmentation results (DeepLabv3+).

For depth training, we use the VisionSim framework [101] (Blender) to generate a dataset of simulated videos with ground-truth depth. The dataset consists of 50 indoor scenes containing ego motion and is rendered at 50 FPS. We randomly select 10 scenes for validation and use the rest for training. See Appendix D.5 for further details on this dataset.

Experiment protocol. For HDRNet and NAFNet, we train spatial-fusion stabilizers using the same settings as in Sections 5.6.1 and 5.6.2. We train HDRNet for the moderate effect strength ($\alpha = 0.5$), and NAFNet for moderate noise ($\sigma = 0.1$) on NFS. Unlike other models, we do not fine-tune the base Depth Anything model; we found that naive fine-tuning on a small dataset like ours quickly led to overfitting. We add controlled stabilizers to instances of DepthAnythingReassembleLayer, DepthAnythingFeatureFusionLayer, and the model output.

		Enhancement		Denoising		Depth estimation		
Corruption	Method	PSNR	Instability	PSNR	Instability	AbsRel (↓)	Delta-1.25 (↑)	Instability
Patch drop	Base model	17.43	164.6	18.93	151.4	0.070	0.948	9.89
	Ours	31.39	30.36	35.46	20.42	0.070	0.956	4.73
Elastic distortion	Base model	24.00	64.23	27.99	47.31	0.052	0.968	7.76
	Ours	26.63	24.97	30.78	21.04	0.057	0.967	4.75
Frame drop	Base model	28.65	94.04	33.42	113.5	0.065	0.936	14.17
	Ours	31.69	29.14	27.34	20.56	0.050	0.974	4.91
JPEG artifacts	Base model	24.85	42.06	29.01	39.71	0.057	0.964	7.32
	Ours	26.46	23.58	32.19	20.49	0.065	0.961	4.92
Impulse noise	Base model	14.28	217.8	24.65	62.10	0.047	0.974	6.83
	Ours	27.37	30.90	32.04	19.73	0.056	0.972	4.98

Table 5.1: **Corruption robustness.** Our method significantly improves stability while preserving or improving prediction quality. The only notable exception is reduced PSNR for denoising with dropped frames. However, we note that PSNR (averaged over frames) does not capture the jarring perceptual effect of dropped frames; thus, the high PSNR of the base model is somewhat deceptive.

We train and evaluate stabilized models for each of the following corruptions: (1) randomly zeroing each 8×8 patch with probability 0.1, (2) elastic deformation, see Appendix D.5 for details, (3) randomly zeroing each frame with probability 0.1, (4) applying JPEG compression at quality 10/100, and (5) adding impulse noise to each channel with probability 0.05 for both salt and pepper. We set $\lambda = 0.2$ when training stabilizers for corruption robustness.

Results. Table 5.1 shows accuracy and stability with and without stabilizers. Adding stabilizers leads to significant reductions in instability and, in most cases, improvements in per-frame accuracy metrics. See Appendix Figures D.5, D.6, and D.7 for qualitative results.

5.6.4 Adverse Weather Robustness

Task, dataset, and base model. We now evaluate robustness under adverse weather conditions. Specifically, we consider the rain and snow corruptions from the RobustSpring [203] dataset. These corruptions differ from those in Section 5.6.3 in their spatial complexity and temporal dynamics (raindrops and snowflakes follow continuous paths, unlike simpler corruptions such as randomly-dropped patches). RobustSpring contains 10 rendered sequences (2000 total frames), each with left- and right-frame variants. Clean ground-truth frames are provided for all sequences. We randomly select 2 videos for validation and use the remaining 8 for training. Videos are downsized to 720p. We evaluate NAFNet denoising [35] with $\sigma = 0.1$, adding noise after weather.

Experiment protocol. We fine-tune the original published weights (with ID nafnet_sidd_width32) with noise, but not weather corruptions. Following Section 5.6.2, we then append a spatial-fusion stabilizer to each NAFBlock and the model output. We train these stabilizers under noise + weather using $\lambda = 0.2$. Consistent with our other experiments, we train stabilizers with a frozen base model.

In addition, we consider two variants with an unfrozen base model. In the first

		Rain			Snow		
Stabilized?	Unfrozen?	PSNR	SSIM	Instability	PSNR	SSIM	Instability
×	×	21.43	0.617	151.76	18.62	0.577	262.48
✓	×	28.63	0.880	57.88	31.34	0.914	59.31
×	✓	32.19	0.937	70.84	34.33	0.950	66.57
✓	✓	32.61	0.938	58.30	35.20	0.956	58.98

Table 5.2: **Adverse weather robustness on RobustSPRING.** Training stabilizers with a frozen base model gives substantial improvement over the unstabilized model. We obtain the best overall results by jointly training stabilizers with the base parameters.

variant, we fine-tune the base model on noise + weather without stabilizers. In the other, we unfreeze the base model when training on noise + weather, jointly training the stabilizer and base parameters. For both variants, we use the same hyperparameters as the frozen stabilizer training.

Results. See Table 5.2 and Appendix Figure D.9 for results. Compared to the unstabilized baseline, stabilizers substantially improve image quality and stability. Likewise, fine-tuning the original weights (without stabilizers) with weather corruptions gives a significant improvement. Compared to stabilization, fine-tuning gives better single-image quality but higher instability. We obtain the best overall results by combining fine-tuning with stabilization. In general, we expect this joint training approach to be the best choice for small- to medium-sized models. For larger models where training the base model is infeasible, training only the stabilizers still gives reasonable results.

5.7 Discussion

Limitations. The bounds in Section 5.4 assume that the distance δ can be expressed in terms of a norm on the prediction space $\mathcal{Y}$. This condition excludes many widely used loss functions, especially more sophisticated, multi-component losses. Nonconforming δ may still work in practice, although they

may require more careful tuning of λ due to the lack of theoretical guarantees.

We had some difficulty with sim-to-real generalization when training stabilizers for Depth Anything. We conjecture that real video contains subtle corruptions not present in simulated video—for example, sensor noise, compression artifacts, or optical phenomena. These "baseline corruptions" could explain some of the temporal instability we see when using apparently clean input video.

Alternate metrics. In all of our experiments, we use a simple Euclidean norm for δ. However, other metrics may be a better choice, depending on the task. For example, a variant of the Wasserstein metric may give improved results for two-dimensional outputs and feature maps, due to its ability to account for the spatial structure of the tensor. See Appendix D.2 for a proposal for such a metric.

6 Discussion

The outlook for generalized event cameras and event-based processing depends greatly on future hardware developments. On the sensing side, the feasibility of generalized event cameras hinges on continued improvements in SPAD sensors and more widespread adoption. Although this outcome is far from certain, recent trends are promising, with major image sensor manufacturers announcing high-resolution SPAD sensors. Further, the compatibility of SPADs with mainstream CMOS manufacturing processes bodes well for scalability and eventual cost.

The future for event-based processing is cloudier. Although event networks are far more compatible with conventional hardware than SNNs, there remains a gap between predicted speedups (reductions in operation counts) and actual time savings. Real runtime depends on many factors—for example, hardware sparsity support, cache sizes, and memory access patterns. A key question influencing the long-term practicality of event networks is whether hardware makers will continue to optimize for a few monolithic operations (e.g., matrix multiplication) or adopt more flexible, sparsity-oriented designs.

As machine learning models have grown in size, especially with the introduction of massive foundation models, it has become increasingly costly for researchers and practitioners to train their own models from scratch. For this reason, lightweight adapter-based approaches are likely to attract more interest. In particular, this bodes well for the stabilization adapter method

proposed in Chapter 5. Unlike event-based processing, this approach provides a more predictable benefit, independent of hardware trends.

A Bandwidth: Generalized Event Cameras

This appendix expands on Chapter 2. Sections A.1 and A.2 provide further details of our methods, Section A.3 contains an extended discussion, and the remaining sections provide expanded experiment details and results.

The supplementary material for our paper [218] contains the following not included here:

- Details of our imaging setup and scene acquisition
- Formal definitions of the proposed event camera algorithms
- Additional high-speed results
- Additional low-light results
- Sample images from the dataset used for rate-distortion experiments
- An illustration of the effect of Ultraphase algorithm modifications on reconstruction quality
- Sample images from the scene used for Ultraphase experiments

A.1 Pipeline Overview

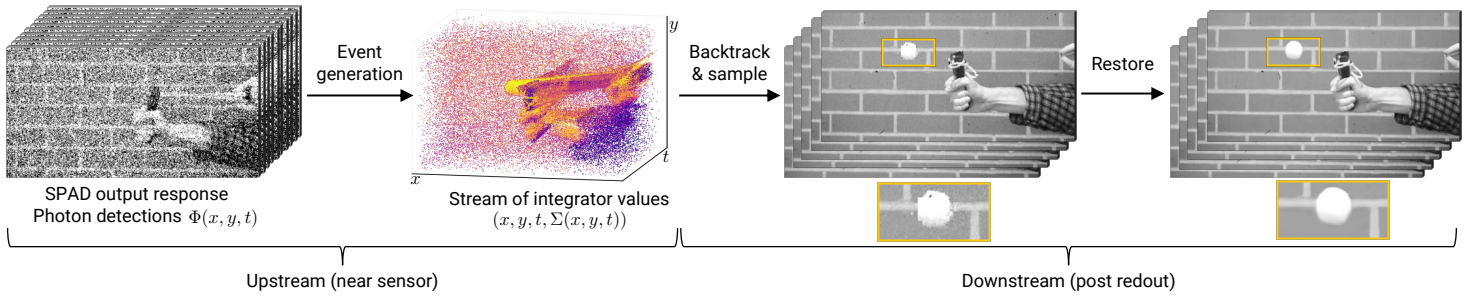


Figure A.1: **Algorithm overview.** Our algorithms take as input the SPAD's response, $\Phi(\mathbf{x}, t)$, and output a stream of integrator values, i.e., a stream of tuples $(\mathbf{x}, t, \Sigma(\mathbf{x}, t))$, where Σ represents the integrator's value. After sensor readout, we perform *backtracking*, which takes the value of the integrator as the flux estimate between the current and previous event timestamps at location $\mathbf{x}$. We then sample these flux estimates at any time t, providing a stack of backtracked frames. Notice the rich scene information present in these backtracked frames. To alleviate artifacts (shown in the insets here) arising from the pixel-wise independent emission of events, we perform video restoration to recover high-quality outputs.

Figure A.1 shows an overview of recovering scene intensity in a bandwidth-efficient manner using generalized event cameras. This process begins from the SPAD's output response, $\Phi(\mathbf{x}, t)$ at pixel location $\mathbf{x}$ and (discrete) time t, and culminates in a high-fidelity, high-temporal resolution video reconstruction. In what follows, we go over the salient steps before laying down specific algorithms (in Sections A.2.1, A.2.2, A.2.3, and A.2.4) and detailing any algorithm-specific modifications to these steps.

Event generation. Each of our algorithms (Sections 2.4, 2.4.1, 2.4.2, and 2.4.3) takes as input the high-speed binary frames produced by the SPAD, $\Phi(\mathbf{x}, t)$, and processes it in an online manner—without any buffering of photon detections. The output of our algorithms is an asynchronous (each pixel or patch can emit events independently) spatiotemporal stream of event packets.

Event packets. For our methods, we assume events are sparsely encoded using a coordinate list (COO) format. In other words, we represent each event as a tuple $(\mathbf{x}, t, \Lambda)$, where $\mathbf{x}$ is the spatial location, t is the time, and Λ is the event payload—which is typically the integrator's value (Σ), except in the case of coded-exposure events where we transmit a set of integrator values. For the adaptive-EMA and Bayesian methods, $\Lambda = \Sigma_{\text{cuml}}$, the adaptive integrator. For the spatiotemporal chunk method, $\Lambda = \mathbf{\Sigma}_{\text{patch}}$, the patch-wise cumulative mean (a vector). For the coded method, $\Lambda = (\Sigma_{\text{long}}, \{\Sigma^{j}_{\text{coded}}\})$, the long exposure and most recent coded exposures (note that we can exclude Σ_{long} if this is the change detector's first time step or if we produced an event on the time step immediately preceding this one).

Backtracking. When an event is emitted at time $t = T_1$, the integrator's value, Σ, is taken to be the flux estimate (or representation) between T_1 and the previous event emission time (T_0)—if this is the first event emission, we assume $T_0 = 0$. Thus, from the spatiotemporal event stream, we can construct a piece-wise constant representation of the incident flux, by traversing the event stream backward in time; we term this process as *backtracking*.

Sampling. After backtracking, we obtain a spatiotemporal cube of intensity estimates. We can now sample this cube discretely to obtain one or more frame-based samples. The main motivation for sampling the intensity cube, rather than processing it in its entirety, is that existing video restoration models are not capable of inference on very long video sequences (most models infer on 32–64 video frames at a time). In this work, we consider a very simple sampling strategy: temporally uniform sampling. There can be, however, more sophisticated ways to sample, potentially based on the rate of events across time [204]—we do not explore these more sophisticated variants in this paper.

Restoration. Having obtained a frame-based sampling (video representation) of our backtracked cube, we can now process it using video restoration techniques. The purpose of video restoration here is to remove artifacts arising from the independence of events between pixels (or spatial patches). For instance, neighboring pixels may fire events at different times, which tends to produce jagged artifacts around motion boundaries. We show these asynchronous artifacts (and their removal) in the insets of Figure A.1.

A.2 Method Details

A.2.1 Adaptive EMA Event Camera

The simplest version of our adaptive exposure technique was introduced in Section 2.3 and further specified in Section 2.4. This technique uses a fixed threshold applied to exponential moving averages (EMAs) to determine changes in scene intensity. Such a choice of change detector is motivated by early works in change-point detection that have utilized exponential moving averages [193]. We show intermediate and final outputs of adaptive-EMA on the slingshot sequence in Figure A.2.

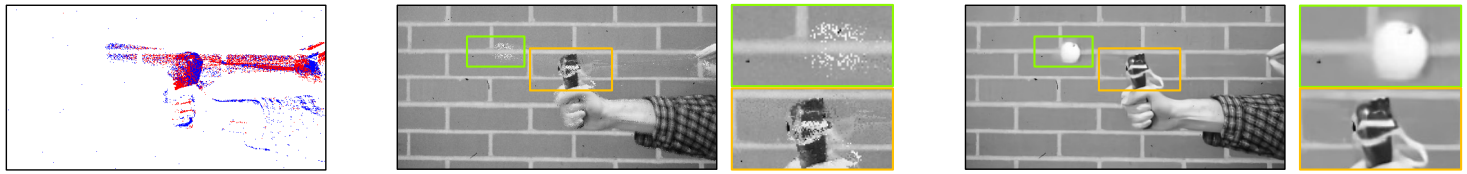


Figure A.2: **Adaptive-EMA example. (left)** We show the changes detected on the slingshot sequence, across 4000 binary frames. **(middle)** The backtracked frame bears noticeable artifacts, **(right)** however this is substantially reduced by merging information from multiple backtracked frames using a video restoration model. Improved reconstructions can be obtained using our more sophisticated methods (Sections 2.4.1, 2.4.2, and 2.4.3) involving more robust change detection procedures.

A.2.2 Adaptive Bayesian Event Camera

Our generalized event camera from Section 2.4.1 uses (a variant of) restarted Bayesian online change detection (R-BOCPD [4]) as the per-pixel change detector. As mentioned in Section 2.4.1, our main modification is the use of *extreme pruning* [240]: instead of storing one forecaster for each time step (or per binary frame), we only retain the top-K forecasters. We found that the performance of the change detector is not significantly impacted even with substantial pruning where we retain just the top-3 forecasters. We now describe a few variants of this algorithm.

Restarts and pseudo-distribution array. We can also consider the base variant of BOCPD [1], which does not involve restarts or an array of pseudo-distribution values. We use this simplification for our on-chip implementation for UltraPhase [7].

Direct extensions to spatial patches. While BOCPD is a per-pixel temporal change detector, there are simple ways to exploit the correlated changes observed in a patch—although these modifications should be seen as simple extensions and not a more principled approach, such as what we describe in Section 2.4.2. To reduce detection delay, we can fire events whenever a changepoint is detected in *any* pixel in a patch. While this may be preemptive for pixels where change has not yet been detected, being preemptive may not be detrimental, as a change in one pixel indicates that changes may soon be observed at other pixels in a patch.

We show intermediate and final outputs of adaptive-Bayesian on the slingshot sequence in Figure A.3. Notice that the dynamics of the slingshot's elastic band and the propelled ping-pong ball are much better preserved in Figure A.3 (as compared to Figure A.2), while entailing a reduced sensor readout as well.

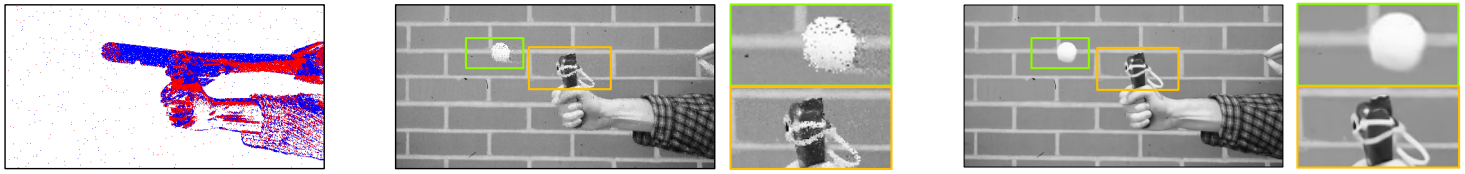


Figure A.3: **Adaptive-Bayesian example. (left to right)** The change points detected by restarted-BOCPD are more informative that those detected by an EMA-based change detector. While there appears to be more change points here than in Figure A.2, the changes are transmitted more parsimoniously over time—as a result, the overall readout is lower (495 bps/pixel for adaptive-Bayesian versus 658 bps/pixel for adaptive-EMA). Using an improved change detector (BOCPD) also results in backtracked images that preserve more detail, and consequently a final reconstruction that has significantly less blur (the ping-pong ball is better recovered here).

A.2.3 Spatiotemporal-Chunk Event Camera

We now describe the spatiotemporal-chunk event camera. Recall from Section 2.4.2 that this method generates an event whenever

$$\|\mathbf{P}(\tilde{\mathbf{\Phi}}_{\text{chunk}}(\mathbf{y},t) - \tilde{\mathbf{\Sigma}}_{\text{patch}}(\mathbf{y},t))\| \geq \tau. \tag{A.1}$$

Let $p \times p$ be the patch size, with $q = p^2$ the number of pixels in a patch. We use a patch size of 4×4 in all our experiments (except for Figure A.4, where we use 8×8 for illustrative purposes). $\tilde{\mathbf{\Phi}}_{\text{chunk}}(\mathbf{y},t) \in \mathbb{R}^q$ is the normalized mean over a temporal chunk of m binary frames; we use $m = 32$ throughout the paper. $\tilde{\mathbf{\Sigma}}_{\text{patch}}(\mathbf{y},t) \in \mathbb{R}^q$ is the normalized cumulative mean since the last event, excluding the current temporal chunk. $\boldsymbol{P} \in \mathbb{R}^{r\times q}$ is a feature matrix. We use $r = 16$ in our main experiments; for UltraPhase experiments, we reduce this to $r = 4$ due to on-chip memory constraints (see Section A.9).

We normalize via

$$\tilde{\mathbf{\Phi}}_{\text{chunk}}(\mathbf{y},t) = \mathbf{\Phi}_{\text{chunk}}(\mathbf{y},t) \oslash \mathbf{c} \tag{A.2}$$

$$\tilde{\mathbf{\Sigma}}_{\text{patch}}(\mathbf{y},t) = \mathbf{\Sigma}_{\text{patch}}(\mathbf{y},t) \oslash \mathbf{c}, \tag{A.3}$$

where $\oslash$ indicates pointwise (Hadamard) division and $\boldsymbol{c}$ is given by

$$\boldsymbol{c}(\mathbf{y},t) = \sqrt{\hat{\boldsymbol{p}}(\mathbf{y},t)\,(1-\hat{\boldsymbol{p}}(\mathbf{y},t))\frac{1}{m}\left(1+\frac{1}{n}\right)}, \tag{A.4}$$

$$\hat{\boldsymbol{p}}(\mathbf{y},t) = \frac{n\mathbf{\Sigma}_{\text{patch}}(\mathbf{y},t) + \mathbf{\Phi}_{\text{chunk}}(\mathbf{y},t)}{n+1}, \tag{A.5}$$

where m is the temporal chunk size, n is the number of temporal chunks comprising $\mathbf{\Sigma}_{\text{patch}}$, and the square root is pointwise. $\boldsymbol{c}^2$ is an empirical estimate of the variance in $\mathbf{\Phi}_{\text{chunk}} - \mathbf{\Sigma}_{\text{patch}}$ under a static assumption, in which case $\mathbf{\Phi}_{\text{chunk}}$ and $\mathbf{\Sigma}_{\text{patch}}$ are binomial random variables with the same probability. We use ghost sampling when computing $\hat{\boldsymbol{p}}$ to prevent numerical instability

near $\hat{p} = 0$ and $\hat{p} = 1$; specifically, we add 8 ghost Bernoulli measurements, 4 of which are successes.

Training matrix $\boldsymbol{P}$. We use a specialized procedure to generate outputs that can be used to train the matrix $\boldsymbol{P}$ via backpropagation through time (BPTT). This procedure does not involve any branched control flow, and gives identical outputs to the base algorithm. The event-generation decision is represented by a binary value k. On the forward pass, k is computed using a Heaviside function. On the backward pass, we approximate the gradients of the Heaviside using a surrogate gradient approach. Specifically, we replace the Heaviside gradients with those of a logistic sigmoid.

Our training procedure also includes an autodiff-compatible backtracking step, where we fill in the values of $\boldsymbol{B}$ in time-reverse order. Unlike an indexing operation, which is not differentiable with respect to its indices, this implementation is differentiable with respect to the values of k, which delimit the piecewise-constant segments of $\boldsymbol{B}$.

We train using interpolated videos from the XVFI dataset [212]. We interpolate from the native 1 kHz frame rate to 16 kHz using RIFE [91], then linearly interpolate over time by another factor of 8 (for a total binary frame rate of 128 kHz). We use the interpolated frames as the ground truth during training. To generate SPAD frames, we treat the ground truth values (in the range [0, 1]) as the Bernoulli photon-detection probability.

We initialize $\boldsymbol{P}$ with uniformly-distributed random values in the range [-0.0125, 0.0125]. We train for 20 epochs, using vanilla SGD with a learning rate of 0.06, reducing the learning rate by a factor of 5 after 10 epochs. Each epoch consists of 200 batches, with each batch containing 64 patch time series. Throughout training, we randomly vary the contrast threshold via uniform sampling in the range [1 / 1.3, 1 / 0.7].

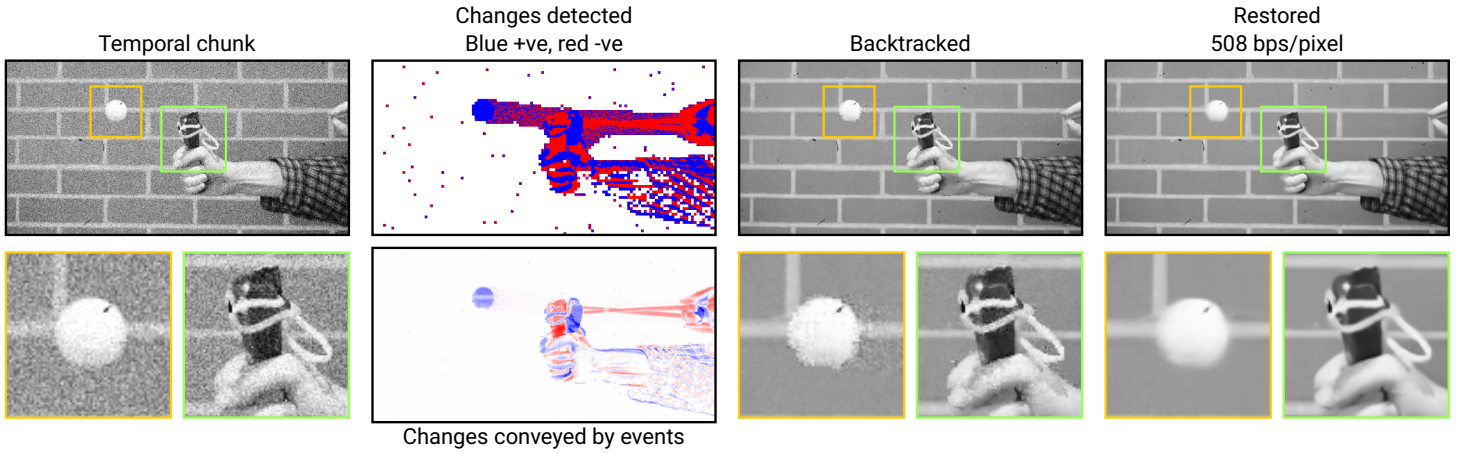


Figure A.4: **Spatiotemporal chunk example. (left to right)** We first accumulate photon measurements into temporal chunks of, e.g., 32 binary frames. We then apply change detection on 4×4 spatial patches. Applying a restoration model to the backtracked outputs removes patch artifacts (e.g., patches that recently triggered an event may appear noisier than their neighbors).

A.2.4 Coded-Exposure Event Camera

Mask pattern (coding) details. We choose $J \geq 2$ random binary mask sequences of length N each, i.e., $\boldsymbol{C}^j \in \{0,1\}^N$. Each of these J sequences is non-overlapping, which is

$$\boldsymbol{C}^j(n)\boldsymbol{C}^k(n) = 0 \;\forall\; j \neq k,\; \forall\; 1 \leq n \leq N. \tag{A.6}$$

The motivation behind such a choice is to ensure that the pseudo-inverse step (Moore-Penrose inverse), which is applied to the coded measurements as a processing step, can be computed efficiently. With these constraints, the pseudo-inverse step involves multiplication (of the coded measurements) by a diagonal matrix, which can be carried out efficiently.

One way to construct such a mask sequence is by choosing $j^* \sim \text{Uniform}(1, J)$ at each sequence location $n \in 1, 2, \ldots, N$, and then setting $\boldsymbol{C}^{j^*}(n) = 1$ and $\boldsymbol{C}^j(n) = 0$ for all other $j \neq j^*$. In other words, we pick a random "bucket" at each subframe index n.

For instance, when $J = 2$, this amounts to picking to mask sequences, $\boldsymbol{C}^1, \boldsymbol{C}^2 \in \{0,1\}^N$, such that

$$\boldsymbol{C}^1(n) = 1 - \boldsymbol{C}^2(n),\; \forall\; 1 \leq n \leq N. \tag{A.7}$$

This choice corresponds to the "coded two-bucket" camera [243]. Thus, these random binary masks can be seen as a generalization of computing a single coded measurement to computing J coded measurements [217].

Finally, we remark that we do not consider the case of $J = 1$ here, since we implement coded exposures *computationally* on single-photon sensors—hence, the "complementary" coded exposure (or the two-bucket measurement) is always readily available. In other words, there is no drastic compute or memory overhead of a (computational) two-bucket coded exposure over a single coded

exposure. This would not be the case if an *optical* setup (e.g., using digital micromirror devices, DMDs) were used to implement coded exposures—where using a two-bucket measurement would likely entail a beam splitter and a second DMD.

Pseudo-inverse step. After backtracking, we can sample a J bucket coded-exposure measurement at any time t. Of course, if the pixel is static, we only get 1 static measurement at the pixel location—so we repeat static measurements J times. Before applying our video restoration module, we perform a pseudo-inverse step that is derived from the linear forward model of J-bucket coded exposures and is similar to the pseudo-inverse pre-processing step adopted for single-bucket compressive captures [264, 256]. This pre-processing step involves computing

$$Y(\mathbf{x}, n) = \sum_{j=1}^{J} \frac{1}{\sum_{m=1}^{N} \boldsymbol{C}^j(\mathbf{x}, m)} \Sigma^j(\mathbf{x}, t) \boldsymbol{C}^j(\mathbf{x}, n), \tag{A.8}$$

giving us N subframes from the J coded measurements.

To summarize, for coded-exposure events, we follow these steps: backtrack and sample → repeat static regions by $J\times$ → pseudo-inverse pre-processing → video restoration. We show intermediate and final outputs of (two-bucket) coded-exposure events on the slingshot sequence in Figure A.5.

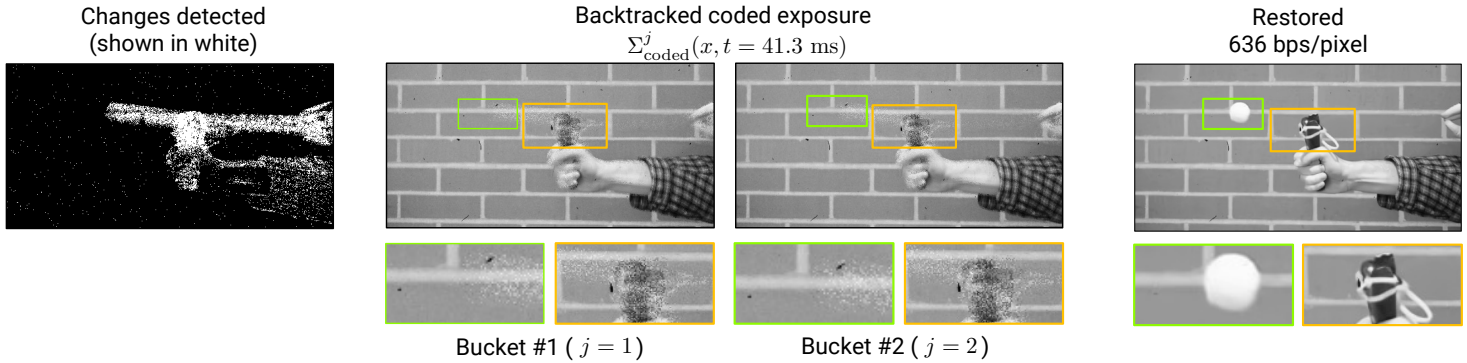


Figure A.5: **Coded exposure (two-bucket) example. (left)** We show the changes detected using the confidence-interval test between two coded measurements that were computed across a temporal chunk of 1000 binary frames. **(middle)** The coded-exposure measurements, Σ^j_{coded}, differ predominantly in dynamic regions, while being statistically similar in static regions—this fact forms the basis of the change detector that we design for coded-exposure events. **(right)** Reconstructions obtained using our video restoration model on a stack of pre-processed and backtracked coded exposures.

A.3 Extended Discussion

A.3.1 More Sophisticated Integrators

In this work, we propose two integrators: adaptive exposures and coded exposures. There remains an extensive, unexplored space of alternate integrators. Both adaptive and coded exposures are linear projections of a pixel's photon detections over time. We could consider more general linear mappings, such as continuous-valued temporal codes. Further, we are not restricted to projections over time; it may also be advantageous to consider spatial projections, e.g., frequency-domain transforms. With these alternate projections, we might be able to reduce bandwidth while maintaining reconstruction quality.

We have so far assumed that our objective is intensity reconstruction. However, there may be situations where we know that the final goal is a specific inference task. In such a scenario, we could design integrators that only encode information relevant to the task at hand; this may be significantly more bandwidth-efficient than transmitting a generic intensity encoding. This integrator could take the form of a learned module, e.g., a neural network, that operates near-sensor and computes a compressed, task-specific scene representation. We could train this module end-to-end with the downstream layers that perform final inference. The resulting system would involve a neural network that spans multiple compute devices, with an event-based communication layer in the middle. This setup somewhat resembles spiking neural networks and other event-based networks, although the goal with such methods is generally reduced computation costs (arising from sparse layer inputs) and not reduced bandwidth along a data-transfer interface.

A.3.2 Entropy Coding and Quantization

Generalized event cameras encode intensity levels, in the range [0, 1], representing the photon-detection rate. In our experiments, we apply uniform quantization to the transmitted values; this keeps our comparisons straightforward and fair.

However, in a practical deployment, it may be more bandwidth-efficient to encode changes rather than values and apply entropy coding to the changes. When transmitting changes, the first event encodes a value in [0, 1], and subsequent events encode a difference in [-1, +1]. This approach is functionally equivalent to transmitting levels or values, assuming the event camera correctly tracks quantization effects (to avoid drift). For natural scenes, the distribution of changes is non-uniform. The shape of this distribution depends in part on the change-detection algorithm; in general, we would expect changes near zero to be unlikely, as these would not trigger an event. We can apply entropy coding (e.g., Huffman coding) to exploit the non-uniformity in the distribution and achieve some additional compression. Entropy coding could give substantial bandwidth savings, albeit at the cost of some increased near-sensor computation.

In addition to entropy coding, we could apply non-uniform quantization, either to values or changes. For example, we could non-uniformly quantize values in [0, 1] based on perceptual considerations (e.g., human sensitivity to intensity differences). With changes, we could exclude portions of the range [-1, +1] based on the characteristics of the change detector; for example, with a fixed contrast threshold τ, there is no need to represent changes in $(-\tau, \tau)$.

A.3.3 Spatial Compression

One advantage of patch-wise events (e.g., as described in Section 2.4.2) is that they permit some spatial compression. For example, we can apply JPEG block

compression to the payload of an 8×8 patch-wise event. Such a change would bring our techniques more in line with existing video-compression algorithms, which compress in both space and time. The compression ratio we observe would depend on the sensor resolution; with higher resolution, we would expect image patches to be more uniform, and thus more easily compressible. Likewise, patches with more noise (e.g., due to a short adaptive integration window) would be more difficult to compress. Under ideal conditions, patch-wise compression might give us an additional $\sim 5\times$ reduction in bitrate. We leave this idea as a topic for future work.

A.3.4 Alternate Sparse Formats

As described in Section A.1, we assume a COO event format $(\mathbf{x}, t, \Lambda)$ when evaluating the bandwidth requirements of our methods. We could improve the sparse format to reduce bandwidth costs further. One such improvement would be to use an implicit time encoding. Specifically, on each binary frame, we would transmit a header (t, n) indicating the current timestamp and the number of events on this frame. We would then send n event packets $(\mathbf{x}, \Lambda)$. If $n \gg 1$, this approach would virtually eliminate the overhead associated with transmitting values of t. Note, however, that this approach assumes we communicate events in time order; i.e., if $t_2 > t_1$, all events at time t_1 are sent before any events at time t_2.

To reduce the overhead associated with $\mathbf{x}$, we could employ sparse matrix formats such as compressed sparse row (CSR) or compressed sparse column (CSC). These formats compress one of the two spatial coordinates (the row and column indices, respectively). The savings relative to COO would depend on the density of events on each frame, with denser events leading to more compression with CSR or CSC.

For pixel-wise methods (i.e., adaptive-EMA, Bayesian, and coded), another potential optimization is to adaptively group events into patch-wise packets.

Assume we are using the COO format $(\mathbf{x}, t, \Lambda)$. If a spatial patch contains many events, it may be more efficient to transmit them in a single packet, as this amortizes the overheads of $\mathbf{x}$ and t. To implement this optimization, we would add an indicator bit b to each event packet. If $b = 0$, then the packet should be treated as a pixel-wise event; if $b = 1$, it is a patch-wise event, meaning Λ encodes integrator information for an entire patch. Within a patch-wise event we would use dense format for Λ, marking the pixels that triggered an event with a one-bit mask. Each patch would adaptively determine whether to encode its events pixel-wise or patch-wise. This decision would be based on the number of events, with the optimal threshold depending on the number of bits required for $\mathbf{x}$, t, and Λ, as well as the patch size.

A.3.5 Latency of the Adaptive Integrator

One limitation of the adaptive integrator $\Sigma_{\text{cumul}}(\mathbf{x}, T_1)$ is that it creates some latency in the intensity estimates. The estimate for the duration $[T_0, T_1]$—which the adaptive integrator computes as the mean of $\Phi(\mathbf{x}, t)$ over $[T_0, T_1]$—is not known until T_1. Thus, there is a latency of $T_1 - t$ to obtain an estimate for $t \in [T_0, T_1]$. The expected delay depends on the frequency of events, with lower delay in more dynamic regions. The latency of $\Sigma_{\text{cumul}}(\mathbf{x}, T_1)$ is not an issue for offline reconstruction and inference (which we assume throughout this paper). However, it may be of concern for certain real-time applications.

As a potential solution, we could introduce a second type of event, which we call an *eager* event, in contrast to the *change* events we have considered thus far. Assume again an event is generated at time T_1. In addition to transmitting $\Sigma_{\text{cumul}}(\mathbf{x}, T_1)$ at that moment, we could send an eager event encoding a noisy estimate of $\Phi(\mathbf{x}, T_1)$. We could then send periodic eager events encoding refinements to the value of the adaptive integrator. Assuming constant flux after T_1, the adaptive integrator converges to the true flux value as time passes; transmitting refinements would allow downstream components to leverage

this improved estimate in the absence of a subsequent change event. We could consider various schedules for sending refinements—e.g., at exponentially increasing intervals, or at fixed intervals until some maximum time has passed. Note, however, that to keep the event camera's bandwidth coupled with (proportional to) the scene dynamics, there would need to be an upper bound to the number of eager refinement events, i.e., with each change event triggering at most N refinements.

The solution described above would involve some additional bandwidth costs. We can thus imagine a generalized event camera that operates in two modes: "online mode" and "offline mode," with eager events only being used in the online mode, where low latency is necessary.

A.4 Restoration Model Details

A.4.1 Model Architecture

While any video restoration model could be used, we choose the densely-connected residual network proposed in Wang et al. [232] (EfficientSCI) for our video restoration architecture—which was successful at restoring backtracked outputs of all of our generalized event cameras (adaptive-EMA, adaptive-Bayesian, spatiotemporal chunk, and coded-exposure events). We also experimented with the spatial-temporal shift-based model of Li et al. [126] (ShiftNet): while this architecture was successful at restoring three of our four proposed events, it did not succeed at restoring backtracked coded exposures. We attribute this to the fact that EfficientSCI was designed with video compressive sensing in mind, whereas ShiftNet is targeted for more general video-restoration tasks.

The memory cost of EfficientSCI scales with the number of input frames. Thus, we are constrained by the device memory in the number of frames we can reconstruct. For example, with 24 GB of GPU memory and a resolution

of 512×256, we can reconstruct about 96 video frames—which can cover a temporal extent of 3000–9000 binary frames (30–90 ms). Increasing the temporal extent requires sampling the backtracked cube at an increased temporal stride, which can lead to blurring and lower-quality results. One solution to this problem might be to employ a recurrent architecture with a fixed-size memory. However, forward-mode recurrent inference must be causal; i.e., the predicted frame at time t may only consider backtracked samples from times before t. An efficient recurrent model may enable reconstructing videos at frame-rates faster than what we show in this work (∼3000 FPS)—indeed, our methods that operate the granularity of individual binary frames (e.g., Section 2.4.1) can, in principle, provide reconstructions at the frame-rate of SPAD photon-detections, i.e., 96.8 kHz.

A.4.2 Dataset

We use 4403 high-speed videos from the training split of the XVFI dataset [212] to train our restoration models. We temporally interpolate these 1000 FPS to 16000 FPS using RIFE [91] and treat each video frame, normalized between 0 and 1 as the photon-detection probability. We then draw 6 binary frames per video frame (as a per-pixel Bernoulli random variable, based on the photon-detection probabilities)—thereby giving us binary-valued responses at 96 kHz. We remark that our dataset generation approach here (unlike in Sections 2.5.3 and A.8) is not physically accurate, since we directly treat the video's values as photon-detection probabilities instead of average photon-arrival rates. We adopt this approach for its simplicity and note that the difference (between detection probabilities and Poisson rates) manifests as a tone-mapping operation by the SPAD's response function ($f(x) = 1-e^{-x}$). We did not see any generalization issues when applying our models trained this synthetic dataset to real SPAD data—in both ambient and low-light scenarios.

A.4.3 Training Parameters

All restoration models were trained until convergence (40–60 epochs) using the Adam optimizer [114] and the mean squared error (MSE) loss objective. We used an initial learning rate of 10^{-5}, which was decayed as per a cosine-annealed scheduler [147] to a minimum value of 10^{-8}. When training the spatiotemporal chunk method, we randomly choose τ on each training iteration from a uniform distribution covering the range [0.76, 1.43]. We also clip the gradient norm to 1.0 to resolve instability during training.

A.5 Baseline Details

EDI++ construction. EDI [171] is a hybrid event-plus-frame technique that can produce a series of sharp images from an event stream and a long exposure spanning the same time duration. There are two practical difficulties in implementing this method: (1) precise spatial- and temporal-alignment is needed between the frames and events, (2) differences in imaging modalities between conventional CMOS cameras and DVS event sensors. There is also a third (DVS specific) difficulty, which Pan et al. [171] overcome by solving an optimization problem (either across one pair of event streams and frames or across multiple such inputs): the contrast threshold in conventional event cameras is not necessarily known and can have inherent randomness [86, 68]. In contrast, when implementing EDI using SPAD-events and frames, we have none of these difficulties: frames and events are perfectly aligned (by construction), the same imaging modality (SPAD photon detection) is used to obtain both events and frames, and finally, the forward (or event generation) model is precisely controlled (no optimization problem has to be solved). Thus, SPAD-based EDI can be thought of as an ideal version of EDI. We further refine EDI outputs using a trained restoration model (with the same architecture used for video restoration of our generalized events), and call this idealized, refined version of EDI "EDI++."

8-bucket compressive sensing. We use the multi-bucket video compressive sensing method described in Sundar et al. [217], with 8 buckets and spanning 32 subframes—where the sum of 64 binary frames comprises a single subframe. For restoring these coded 8-bucket capture, we use EfficientSCI [232], while retaining the same densely-connected residual architecture that constitutes the video restoration model for the proposed generalized events.

Burst denoising baseline. We perform burst denoising on a stack of 32 short exposures—where the sum of 64 binary frames constitutes one short exposure—using the align-and-merge technique of Hasinoff et al. [76]. After the merging step, we additionally perform BM4D denoising [155] (with $\sigma = 0.02$, after binomial variance stabilization [263]). Further, we find that BM4D applied directly to the stack of short exposures (with binomial variance stabilization) results in poor performance ($\sim$28 dB PSNR on the rate-distortion plot of Figure 2.8).

A.6 Camera Motion Experiments

In this subsection, we provide a qualitative analysis of the impact of ego-motion on the event-generation rate and the output-image quality. We consider three scenes with camera motion: the "building" sequence that features a significant amount of spatial structure and image detail (downtown buildings); the "Ramanujan bust" sequence, which is an indoor scene with a moderate amount of texture (from the bust and metal plaque); and a nighttime driving sequence where the SPAD was placed on the car's dashboard.

We show the change points and reconstructions obtained using one of our proposed generalized events (adaptive-Bayesian, Section 2.4.1) for these sequences in Figures A.6, A.7, and A.8. To bring out the effect of camera motion, we speed up the SPAD's output response (by skipping binary frames) by factors of 2×, 4× and 8×. Thus, a 4× sped up sequence sees 4× as much camera

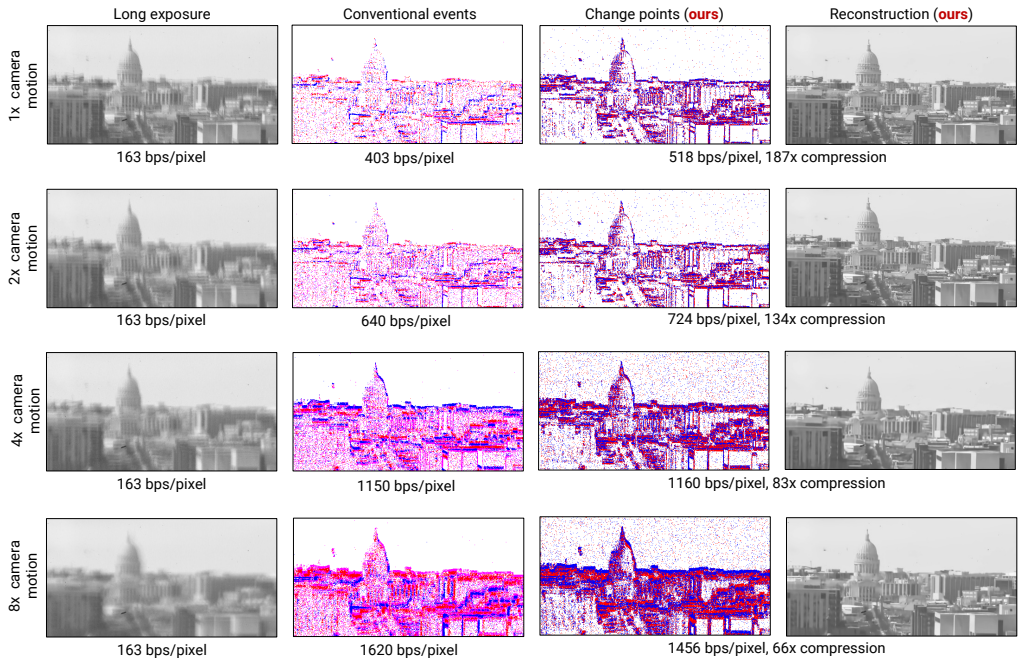


Figure A.6: **Ego-motion on the "building" sequence.** We run our generalized event technique on 8000 binary frames in each row. In the second to fourth rows, we speed up photon-detections by processing only every n-th binary frame (n =2, 4, 8)—this exaggerates ego-motion. **(left to right columns)** We depict the extent of motion using a long exposure across the same duration. We also show the changes detected by a (SPAD-based) event camera [217]. With increasing ego-motion, our techniques output more change points (as expected), but the reconstructions remain relatively sharp (there is some amount of blur induced), even with 8× more motion.

motion. For additional context, we also show the extent of motion using a long exposure and the response of a (SPAD-based) event camera across the same duration.

Across all sequences, we observe that even with an exaggerated amount of camera motion (e.g., 4× and 8× sped-up sequences), we still see a significant amount of compression (reported with respect to raw photon readout) and a modest sensor readout (reported in bps/pixel).

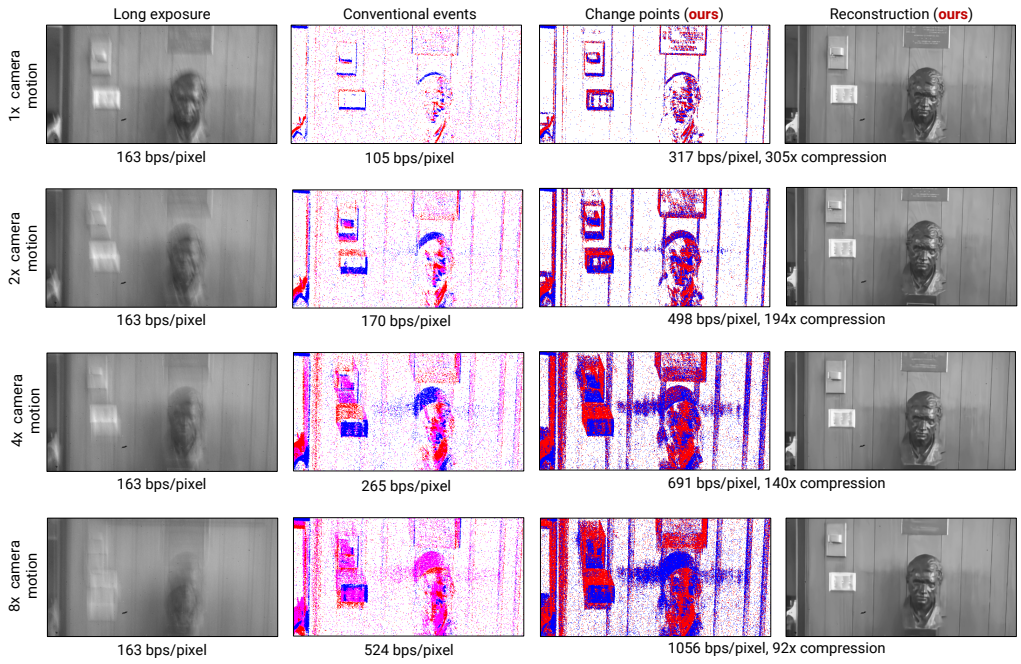


Figure A.7: **Ego-motion on the "Ramanujan" sequence.** The sequence comprises 8000 binary frames, each acquired at a frame rate of 96.8 kHz. Column and row descriptions are identical to Figure A.6. This sequence has comparatively less texture than the building sequence, consequently, the blur in our output reconstructions (at 8× the camera motion) is far less perceptible.

A.7 Plug-and-Play Event Inference

Expanded results. Figure A.9 shows an expanded set of plug-and-play inference results. This figure includes results for a 42 ms long exposure (4096 binary frames, second row) and a burst reconstruction method [151], run on consecutive 0.3 ms short exposures (32 binary frames). The long exposure fails to capture fast-moving objects; there is significant blur in the arm, racket, and ball, causing inference to fail in these regions. The burst reconstruction gives good-quality inference in both static and dynamic regions; however, it requires a high readout bandwidth (about 30× higher than our method).

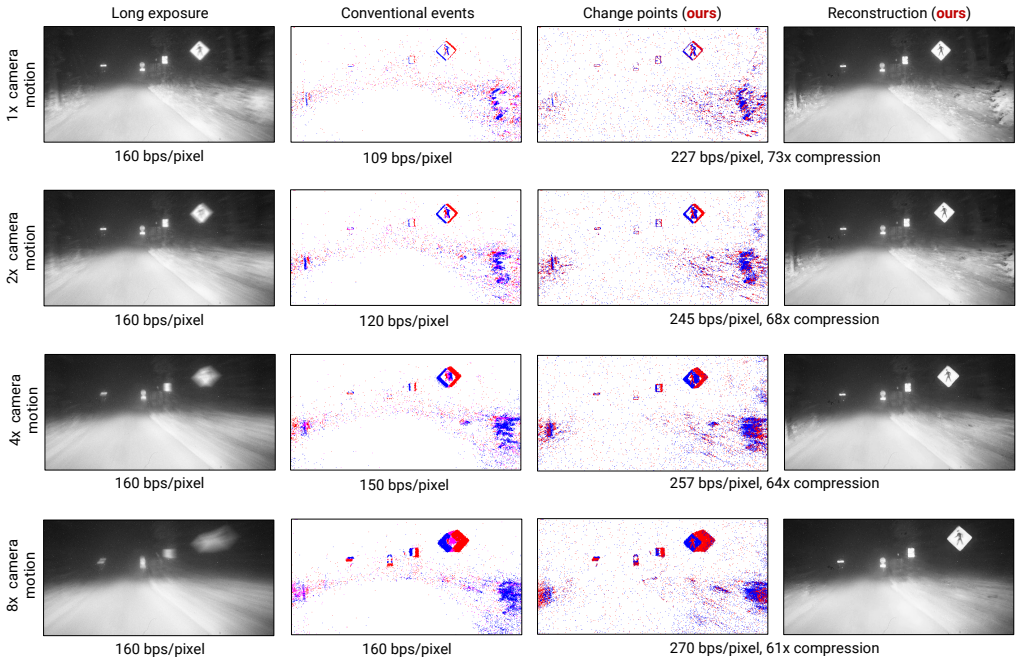


Figure A.8: **Ego-motion on the "night driving" sequence.** Column and row descriptions are identical to Figure A.6. The SPAD was operated at a lower speed in this sequence (16.6 kHz instead of 96.8 kHz)—we report compression factors with respect to photon-detection readout at 16.6 kHz. While this sequence also has lesser texture than Figure A.6, the low-light conditions here make it more challenging. When ego-motion is exaggerated by 4× (or higher, last two rows), we observe that details of the bushes are blurred out. However, the pedestrian crossing sign, which is has better contrast, is still recovered.

Experiment details. We manually trim the Prophesee outputs to a temporal extent corresponding to the SPAD capture (consisting of 8192 binary frames). We run a Prophesee-provided E2VID model on these events to reconstruct a video. For our method and the burst reconstruction in Figure A.9, we run pose detection, corner detection, object detection, and segmentation on the reconstructed frame corresponding to the 2224^{th} binary frame. For the long exposure results in Figure A.9, we run these methods on the mean over binary frames 0–4095. See below for optical flow extents. Below we provide additional details for some of the experiments in Figures 2.7 and A.9.

- **HRNet pose:** We use the HRNet-W48 version of the model.
- **Harris corners:** We run a standard Harris corner detector with $\sigma = 4$.
- **RAFT flow:** For our method and the burst reconstruction in Figure A.9, we run RAFT between the reconstructions corresponding to the 2224^{th} and 2864^{th} binary frames. For the long exposure results in Figure A.9, we consider the interval between the 0–4095 and 4096–8191 exposures. We use the RAFT-Large version of the model.
- **DETR detection:** In most cases, we use a confidence threshold of 90%. We lower the threshold to 80% for the E2VID reconstruction given the lack of high-confidence predictions. We use the ResNet-50 version of the model.
- **Arc* corners:** We run the algorithm on the entire event sequence. We temporally trim the predicted corners to a 0.8 ms window around the target instant. In the figure, we show events within an 8 ms window to provide visual context.
- **E-RAFT flow:** We prepossess events into a voxel grid with 30 bins (divided into 2 sub-durations), with a time span corresponding to that used in the RAFT experiments.

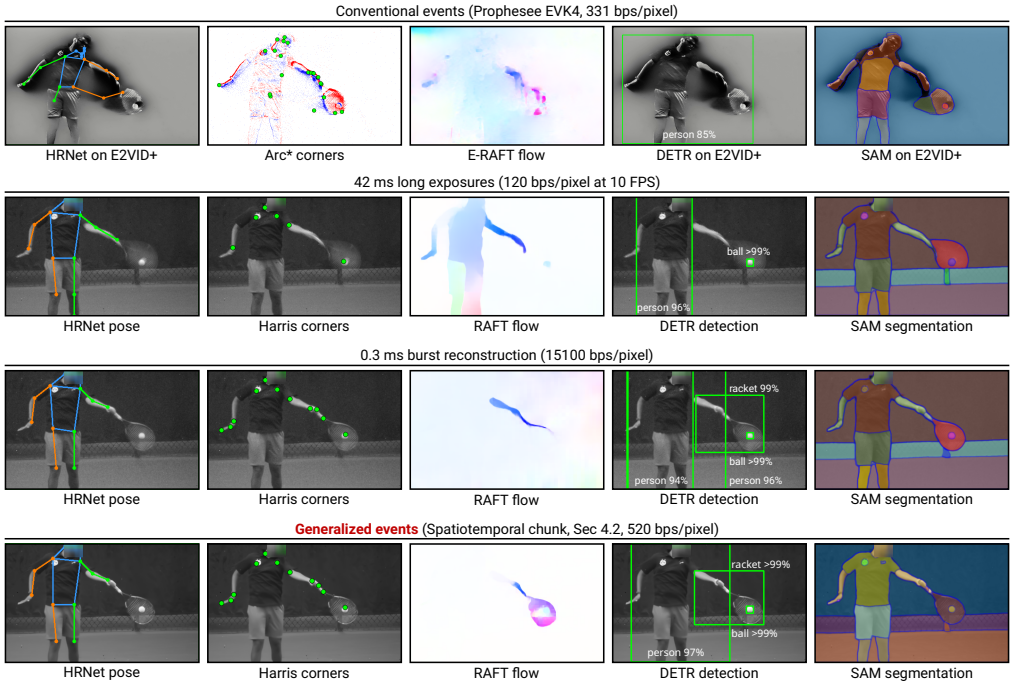


Figure A.9: **Expanded plug-and-play results.** In addition to the results shown in Figure 2.7, we show results for long exposures (consisting of 4096 binary frames) and burst reconstructions [151].

A.8 Rate-Distortion Evaluation

Dataset details. We source 15 videos from YouTube that were captured by a Phantom Flex 4K camera at 1000 FPS. See the supplement of our paper [218] for thumbnails depicting the scene content in each video. We download these videos at a resolution of 854×480 pixels and further downsize (by 1.6×) and vertically crop them to the SPAD's resolution of 512×256 pixels.

We did not utilize the XVFI dataset [212], which we used for training our video restoration models, for evaluating rate-distortion tradeoffs to prevent the possibility of data leakage. Further, we find that these YouTube-sourced videos have a more extreme range of motion than XVFI videos.

Simulating SPAD responses. To simulate SPAD photon detections from high-speed videos, we adopt the following steps:

1. We interpolate videos at 1000 FPS by 16× using RIFE [91] to 16000 FPS.
2. We treat the videos as sRGB-encoded frames, convert them to linear RGB images, and then grayscale.
3. Next, we determine the average count of incident photo-electrons as

$$N(\mathbf{x}, t) = \alpha \mathcal{I}(\mathbf{x}, t) + d, \tag{A.9}$$

   where we $\mathcal{I}(\mathbf{x}, t)$ represents a video frame, d is number of spurious detections ((7.74×10^{-4}) counts per binary frame, using values reported in Ulku et al. [224]). We choose α such that the average value of $\alpha\mathcal{I}(\mathbf{x}, t)$ over each pixel location $\mathbf{x}$ and time t is 1—we find that our ambient captures with the SwissSPAD2 have an average photon per pixel per binary frame (PPP) of 1.
4. Finally, we draw binary frames with the probability of a 1 given by

$$\text{Prob}\{\Phi(\mathbf{x}, t) = 1\} = 1 - e^{-N(\mathbf{x},t)}. \tag{A.10}$$

   From each frame at 16000 FPS, we draw 6 binary frames, thereby simulating photon detections at 96000 Hz.

Computing metrics. We evaluate perceptual distortion using PSNR (computed from the average mean squared error across the entire video [109]), SSIM (computed per-frame and averaged) and MS-SSIM (computed per-frame and averaged) metrics. Both metrics are converted with respect to linear values. Specifically, each of our generalized event cameras and baselines (EDI++, burst denoising, coded 8-bucket) recovers the time-varying estimate of $1 - e^{-N(\mathbf{x},t)}$, i.e., the probability of photon detection. Let us denote this by

$\hat{p}(\mathbf{x}, t)$. We can obtain linear estimates by computing

$$\frac{1}{\alpha}\left(\log\left(\frac{1}{1-\hat{p}(\mathbf{x},t)}\right)-d\right). \tag{A.11}$$

Baseline parameter sweeps. We implement EDI++ with SPAD events, with an exponential decay of 0.95, and sweep the threshold between 0.3–0.54. The other baselines (burst denoising, long exposure, compressive sensing) are frame-based, and do not feature a tunable parameter that controls their readout rate.

Generalized-event parameter sweeps. For adaptive-EMA, we set the exponential decay (of the EMA) to 0.95, and sweep the threshold between 0.3–0.54. For adaptive-Bayesian (Section 2.4.1), we retain the top-3 forecasters and vary the sensitivity γ of BOCPD uniformly (on a logarithmic scale) between 10^{-7} and $10^{-2.2}$. For the spatiotemporal chunk method (Section 2.4.2), we use a patch size of 4×4 pixels, average 32 binary frames per temporal chunk, and vary the threshold (τ) of the change detector uniformly between 0.6 to 1.28. For coded-exposure events, we use a chunk size of 1024 binary frames, with 4 coded buckets (measurements per chunk) that multiplex 16 subframes each—thus, each subframe consists of $1024/16 = 64$ binary frames. We vary the confidence level of Wilson's score ("α" parameter) uniformly within 1.2–6.8.

Extended results. In addition to the PSNR-based rate-distortion plot shown in Figure 2.8, we include evaluations based on SSIM and MS-SSIM metrics in Figure A.10 (top). Across metrics, we see that generalized event cameras provide a pronounced difference in performance over the considered baselines (shown in shades of gray). (bottom) We include another evaluation on 4096 binary frames (instead of 2048). We observe that the readout rate, measured as bps/pixel, is lower across this extended duration—since the fixed readout costs of static (and less dynamic) regions is amortized over a longer duration, unlike frame-based cameras that involve fixed readout for all regions of an

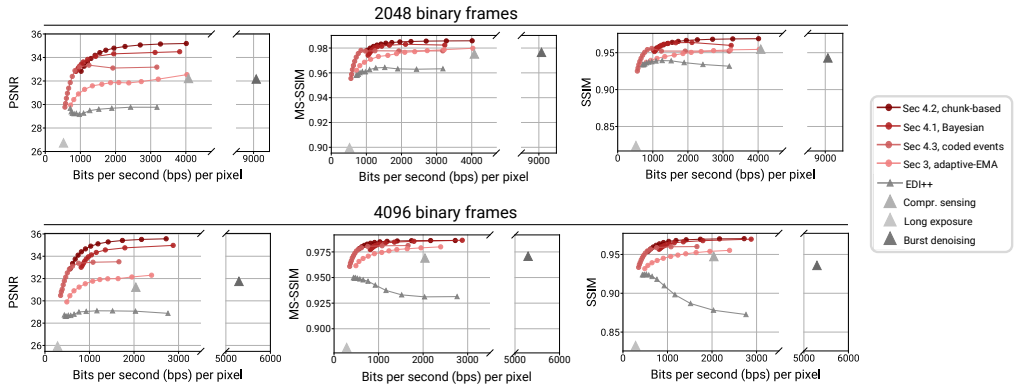


Figure A.10: **Extended rate-distortion evaluation. (top)** Rate-distortion evaluations based on PSNR, SSIM and MS-SSIM metrics across 2048 binary frames. **(bottom)** When considering a longer temporal extent, i.e., 4096 binary frames, we see that the readout rate at which a significant PSNR (or other metrics) drop-off is noticed becomes smaller—in other words, we obtain more compression of raw photon detections.

image. Finally, we remark that the performance gap between EDI++ and our techniques is further widened across this longer duration.

A.9 UltraPhase Experiments

System description. UltraPhase consists of 3×6 cores, each of which processes data from a 4×4 patch of SPAD pixels. The chip operates at a frequency of 0.42 GHz, implying a maximum of 4341 instructions per binary frame at 96.8 kHz. We additionally refer readers to Ardelean [7] for a detailed description of the chip architecture.

Implementing event cameras on UltraPhase. We implement our event camera designs in UltraPhase assembly code. At this point, UltraPhase does not have native hardware for computing divisions—so we modify our methods to work avoid division operations. For the Bayesian method, we skip restarts and avoid division when computing the likelihood by multiplying

arguments with their least common multiple—this is possible because the min and argmax operations carried out in BOCPD are invariant to the scale (of forecaster values). For the spatiotemporal chunk method, we skip the normalization step and use a 4×16, 8-bit quantized feature matrix $\boldsymbol{P}$, in contrast to the 16×16 matrix we use in the rest of our experiments. For the coded-exposure method, we consider (2-bucket, 8-subframe) masks; additionally, we replace Wilson's score by a fixed confidence interval (essentially amounting to a fixed threshold operation). For most scenes, the above modifications do not significantly reduce the quality of the results. See the supplement of our paper [218] for a comparison.

Clock cycle measurements. Due to circumstances beyond our control, we were unable to run our methods on a physical testbed system. We evaluate the runtime characteristics of our methods by assembling them for UltraPhase and measuring the number of compute cycles required to execute them. Given the deterministic nature of the digital hardware, the compute and memory requirements we measure in this evaluation are identical to those we would measure in physical hardware.

We confirm that all methods operate within the memory budget of the chip. In Table A.1, we show the measured clock cycles (the average per binary frame) for each method. In the case of branching, we assume the more computationally expensive branch is taken. Therefore, all compute values are an upper bound.

Readout estimation. The chip readout depends on the dynamics of the scene. To estimate the readout, we run our methods on a 12×24 crop from the tennis sequence in Figure 2.7, over 2500 binary frames. We show frames from this crop in the supplement of our paper [218]. We scale the measured readout values to units of kilobytes per second; see Table A.1 for results.

Power estimation. We estimate two components of power consumption: compute power and chip readout power. We base our analysis on [217]. We

assume the chip consumes 3.5 picojoules per clock cycle spent executing an instruction, and that the chip expends 54 nanowatts per kilobit of readout. Table A.1 shows our estimated compute and readout power.

Table A.1: **UltraPhase results.** We measure the number of compute cycles (per binary frame) required to implement each of our methods, and estimate the readout bandwidth. Based on these values, we estimate the power required for on-chip computation and readout. All of our methods fit within the chip's computational budget of 4341 instructions per binary frame and give two orders of magnitude reduction in bandwidth compared to reading out raw photon detections. Due to these bandwidth reductions, our methods are also much more power efficient than raw photon readout.

Method	Cycles	Bandwidth estimate (kB/s)	Compute power (W)	Readout power (W)	Total power (W)
Raw photons	12	3600	5.43×10^{-9}	1.57×10^{-3}	1.57×10^{-3}
Adaptive-EMA	2367	59.0	2.11×10^{-4}	2.57×10^{-5}	2.37×10^{-4}
Bayesian	3095	57.6	3.62×10^{-4}	2.51×10^{-5}	3.87×10^{-4}
Spatiotemporal chunk	380	53.3	5.45×10^{-6}	2.32×10^{-5}	2.87×10^{-5}
Coded	177	33.1	1.18×10^{-6}	1.44×10^{-5}	1.56×10^{-5}

B Compute: Event Neural Networks

In this appendix we provide further details, results, and discussion for the Event Neural Networks work in Chapter 3. The supplementary material of our paper [50] contains additional result samples and complete result tables.

B.1 Results on Low-Level Tasks

In this section, we describe our experiments for low-level tasks. We consider HDRNet [64] for image enhancement and PWC-Net [215] for optical flow.

Note that these models include some specialized operations (i.e., the bilateral transform for HDRNet [64] and flow warping in PWC-Net [215]). These operations represent a small portion of the overall computational cost of the models. For simplicity, we exclude them when counting MAC operations.

Image enhancement. HDRNet [64] can be trained to reproduce several image enhancement effects. We use the Local Laplacian [178] version of the model. HDRNet has two sub-networks: a deep, low-resolution feature network and a shallow, high-resolution guidemap network. The guidemap network represents only about 10% of the overall operations, and converting it to an EvNet has a noticeable effect on the visual quality of the output. Therefore, we only convert the feature network to an EvNet. We report operation

Table B.1: **Results on low-level tasks.** Results for image enhancement and optical flow. The tables gives the overall computation savings, agreement between the conventional and event models, and overhead percentages (number of extra operations expended for each operation saved).

Model	Savings	Agreement	Math	Load/Store
HDRNet-a	5.78×	39.4 PSNR	2.57%	3.79%
HDRNet-f	23.9×	39.4 PSNR	2.57%	3.79%
PWC-Net	2.68×	0.335 EPE	0.44%	0.74%

savings for both the overall model (both sub-networks) and the feature network (the EvNet portion). We refer to these operation counts as "HDRNet-a" and "HDRNet-f," respectively. We use a threshold of $h = 0.1$ and evaluate the PSNR metric. We resize all images to 540×960 before applying the model.

We use the original authors' pretrained weights. However, these weights were trained on a non-public dataset. Therefore, instead of evaluating the model against ground truth labels, we compute the agreement between the outputs of the event model and conventional model. We evaluate on a subset of the MPII video dataset [6] (see Section B.4 for details on the dataset). See Table B.1 and Table B.2 for results.

Optical flow. We also consider the PWC-Net model [215] for optical flow computation. Unlike the other models (OpenPose, YOLO, HDRNet) which take a single frame as input, this model takes a *pair* of frames. We use a threshold of $h = 0.01$ and evaluate using the EPE metric [10]. We resize all images to 288×512 before applying the model. We use the original authors' weights trained on Sintel [22]. Like with HDRNet, we evaluate the agreement between the event and conventional outputs. Results are shown in Table B.1 and Table B.2. We also evaluate on the ground-truth labels in the Sintel training dataset. On this data, the conventional model achieves EPE 2.86 and the event model achieves EPE 3.33.

Table B.2: **Camera motion for low-level tasks.** The savings factor for different levels of camera motion, evaluated on our custom MPII dataset (see Section B.4).

Model	None	Minor	Major
HDRNet-a	6.19×	6.02×	5.55×
HDRNet-f	34.9×	29.7×	20.0×
PWC-Net	5.41×	3.29×	2.11×

B.2 Additional Analysis Experiments

Varying granularity. Table B.3 shows the effect of increasing the granularity of the policy. We evaluate the OpenPose model [28] on the JHMDB dataset [100]. We test both a spatial chunking policy and a policy that chunks along the channel dimension (e.g., [73]). Because each neighborhood computes a mean of several $|d|$, the thresholds must be reduced to keep the accuracy from dropping. The threshold-setting strategy $0.05/\sqrt{n}$ is a heuristic that we found to give relatively stable accuracy with varying n. The results show that increasing the chunk size reduces the operation savings. However, chunking may, in practice, allow more efficient execution on some hardware.

Comparison against output interpolation. One alternate strategy for efficient video inference is to run a model once every n frames and interpolate its predictions for the remaining $n-1$ frames. We apply this strategy to OpenPose on JHMDB and compare it to the EvNet approach. We use n =16 and linearly interpolate the joint positions between model predictions (the value 16 was chosen to give a computational cost close to the EvNet in Table 3.1). The interpolated model expends 6.764×10^9 ops per frame on average and achieves a PCK of 68.52% (a reduction of 7.88% from the conventional model). Compare this to the EvNet in Table 3.1, which expends 6.780×10^9 ops on average while achieving a PCK score of 76.37% (a reduction of 0.03% from the conventional model). Compared to output interpolation, the EvNet gives much higher accuracy at a similar computation cost. Note that we trim the inputs for the

Table B.3: **Varying granularity.** Results for the OpenPose model on the JHMDB dataset. Using larger chunks, especially channel chunks, reduces the computational gains somewhat. However, chunking may have practical benefits on some hardware.

Variant	Threshold	PCK	Operations
Conventional	—	0.7640	7.055×10^{10}
No chunking	0.05	0.7581	6.166×10^{9}
2×2 chunks	$0.05/\sqrt{2}$	0.7575	8.574×10^{9}
4×4 chunks	$0.05/\sqrt{4}$	0.7662	1.191×10^{10}
8×8 chunks	$0.05/\sqrt{8}$	0.7431	1.646×10^{10}
Channel chunks	0.02	0.7600	1.782×10^{10}

interpolation model to have a length of $kn + 1$ frames, where k is a positive integer. This ensures that the video can be divided into uniform blocks of n frames (with one extra frame at the end). If we trim to the same length for the EvNet, it achieves a PCK of 76.82% with average cost 7.265×10^{9} ops. The conventional model here yields PCK 76.67% at 7.055×10^{10} ops.

Temporal smoothness. We have anecdotally observed improved temporal smoothness in the outputs of EvNets. We hypothesize that this is one of the reasons for the slightly increased accuracy for some models (e.g., Table 3.1) over the conventional baselines. We quantitatively measure temporal smoothness for OpenPose on JHMDB by measuring the mean L2 joint motion between frames. The average joint motion for the conventional model is 10.3 pixels. For the EvNet with threshold $h = \{\ 0.01, 0.02, 0.04, 0.06, 0.08\ \}$, the average motion was $\{\ 9.77, 9.26, 7.98, 7.14, 5.81\ \}$ pixels. This confirms that the EvNet outputs are more temporally smooth than those of the conventional model, with smoothness increasing with the policy threshold.

B.3 HRNet Experiments

We test HRNet [216], a state-of-the-art model for various location-based tasks (e.g., object detection) on the JHMDB pose recognition dataset [100]. We use

Table B.4: **HRNet Results.** Results for the HRNet model on JHMDB.

Model	Threshold	PCK	Operations
Conventional	—	90.37%	1.019×10^{10}
EvNet	0.05	90.43%	1.112×10^{9}
EvNet	0.1	90.46%	7.361×10^{8}
EvNet	0.2	86.44%	4.187×10^{8}
Skip-Conv	0.05	89.17%	1.035×10^{9}
Skip-Conv	0.1	84.45%	6.473×10^{8}
Skip-Conv	0.2	78.72%	3.307×10^{8}

the HRNet-W32 version of the model.

Training procedure. We initialize with pretrained MPII weights from [216]. We fine-tune the model on JHMDB for 20 epochs using the Adam optimizer and a learning rate of 10^{-5}. We set aside 20% of the training data for validation and save the model at the epoch with the lowest validation loss. JHMDB defines three train/test splits—we train and evaluate a model on each training split and average the results (accuracy and computation costs) over the three splits. Where not otherwise specified, we adopt the training and data augmentation parameters of [73]. All of our training code will be publicly released and included with the supplementary material.

Evaluation. We evaluate three model variants: the conventional model, an EvNet and Skip-Conv (without periodic resets). See Table B.4 for results. We report the PCK metric (as in our experiments on OpenPose). The accuracy and savings we observe are in line with our other experiments.

B.4 Experiment Details

Custom MPII dataset. Here we describe the dataset that we use in our camera motion experiments (Table B.2) and for evaluating HDRNet and PWC-Net (Table B.1). We take a subset of the MPII video dataset [6]—specifically, the first 246 videos that have exactly 41 frames (most, but not all videos in MPII have 41 frames). We then label each video in this dataset as having “no

camera motion" (perfectly stationary camera), "minor camera motion" (slight camera shake), or "major camera motion." These splits contain 59, 46, and 141 videos, respectively.

Overhead counting. We count overhead operations as follows. An update to an accumulator requires one load (a), one addition ($a + g(\boldsymbol{\Delta}_{\text{in}})$), and one store ($a$). An update to a gate requires two loads (b and d), three additions ($d + f(a) - b$ and $|d| - h$), and two stores (b and d). A transmission requires one load (d), one subtraction ($d - \Delta_{\text{out}}$), and one store (d).

B.5 Derivation of Equation 3.4

The equation for $a^{(T)}$ is a consequence of the update to $a^{(t)}$ defined in Equation 3.3, combined with the linearity of g (g of a sum is equal to the sum of the g). The equation for $b^{(T)}$ is a direct consequence of the update rule in Equation 3.3.

The equation for $d^{(T)}$ in Equation 3.4 comes from combining Equation 3.3 and the post-transmission subtraction of Δ_{out}. Let $d^{(0)} = 0$ as stated in Section 3.4.2. With Equation 3.3, and noting that $b^{(t)} = f(a^{(t)})$,

$$d^{(T)} = \sum_{t=1}^{T} \left(f(a^{(t)}) - b^{(t-1)} \right) = b^{(T)} - b^{(0)}. \tag{B.1}$$

Adding in the post-transmission subtraction of $\Delta_{\text{out}}^{(t)}$, we have

$$d^{(T)} = b^{(T)} - b^{(0)} - \sum_{t=1}^{T} \Delta_{\text{out}}^{(t)}. \tag{B.2}$$

B.6 Thoughts on Theoretical Guarantees

For certain special cases of transmission policies (e.g., a threshold policy with $h = 0$), we can guarantee that the output of an EvNet will be equal to that of the equivalent conventional network. As we make the policy more selective (e.g., by increasing h), the efficiency of the EvNet improves, but its output increasingly deviates from that of the conventional network. While we currently describe this behavior qualitatively, developing the theoretical tools necessary for a quantitative description is an important next step.

We can describe a neural network as a composition of functions,

$$\boldsymbol{y} = q_m(\ldots(q_2(q_1(\boldsymbol{x})))\ldots). \tag{B.3}$$

We can think of an event network as perturbing the output of each q_i by some ϵ_i. That is,

$$\boldsymbol{y}' = q_m(\ldots(q_2(q_1(x) + \epsilon_1) + \epsilon_2)\ldots) + \epsilon_m. \tag{B.4}$$

If we assume a threshold policy with threshold h, then $||\epsilon_i||_\infty < h$. Given these facts and some knowledge of the properties of the q_i (e.g., the distribution of their weights), can we bound the norm of $\boldsymbol{y} - \boldsymbol{y}'$? This question has important implications for applications that require accuracy guarantees and should be studied in future work.

C Compute: Eventful Transformers

This appendix expands on Chapter 4. The supplementary material for our paper [51] contains additional result tables not included here.

C.1 Further Discussion

The ViViT temporal sub-model. Recall that, for ViViT action recognition, we fine-tune the non-Eventful temporal model on the outputs of the Eventful spatial model. We now provide some intuition as to why this is necessary.

The outputs of an Eventful Transformer are approximations of the "correct" outputs (those of the original, non-Eventful Transformer). In the case of the ViViT spatial model, individual outputs are fairly close to the correct values. However, the *pattern of temporal changes* between outputs may be quite different from the original model. Token gates reduce the number of updated tokens on each frame, but each update tends to be larger (a single update may contain accumulated changes from several time steps). Given the nature of the prediction task—action recognition on highly dynamic videos—the temporal sub-model is sensitive to the pattern of temporal changes. Fine-tuning allows us to correct for the shifts in these temporal changes that result from using an Eventful spatial model.

Compatibility with spatial redundancy methods. We now provide further discussion regarding the compatibility of our method with spatial redundancy approaches. Abstractly, we can think of spatial redundancy methods as summarizing a set of tokens $\boldsymbol{x} \in \mathbb{R}^{N \times D}$ using a reduced set of tokens $\hat{\boldsymbol{x}} \in \mathbb{R}^{M \times D}$. The simple method in our experiments summarizes tokens using uniform pooling; however, we could also use adaptive pruning or merging.

Assume we apply a gate to the reduced tokens $\hat{\boldsymbol{x}}$. The gate assumes that the definitions of its input tokens are relatively stable. This assumption clearly holds for non-reduced or uniformly pooled tokens. However, we need to be careful when applying arbitrary reductions to $\boldsymbol{x}$.

For example, say we have an image containing a region of blue sky. An adaptive token merging method might combine all sky-colored tokens from $\boldsymbol{x}$ into a single token in $\hat{\boldsymbol{x}}$. Assume that on frame $t = 1$, the first token in $\hat{\boldsymbol{x}}$ represents the sky. Ideally, on frame $t = 2$, the first token in $\hat{\boldsymbol{x}}$ should again represent the sky. Note that this is not a strict constraint—our gating logic can deal with non-consistent definitions for a few tokens. However, if the definitions for all tokens in $\hat{\boldsymbol{x}}$ completely change between frames, then the gate will not be able to keep up (i.e., the number of tokens with significant changes will exceed the policy r-value).

C.2 Additional Experiments

Video action recognition on Kinetics-400. We evaluate our method on the Kinetics-400 action recognition dataset [107]. Kinetics-400 contains over 300k video clips, each annotated with one of 400 action categories. We evaluate top-1 accuracy. We use the same ViViT model architecture as in our EPIC-Kitchens experiments; the only difference is the input size (224×224 rather than 320×320).

As in our EPIC-Kitchens experiments, we fine-tune the non-Eventful tempo-

Table C.1: **Kinetics-400 video action recognition.** Results for Kinetics-400 action recognition using the ViViT model. We report the total TFlops per video (spatial + temporal sub-models).

Variant	r	Accuracy (%)	TFlops
Base model	—	79.06	3.360
Temporal	96	77.62	1.814
Temporal	48	75.88	1.016
Temporal	24	75.16	0.618

ral model on the outputs of the Eventful spatial model. We fine-tune three variants of the model with r =24, 48, and 96 (out of a maximum of 197 tokens). We train for 10 epochs on a subset of the training set containing 39729 videos. We use the AdamW optimizer [148] with a learning rate of 2×10^{-6}, weight decay of 0.05, and a batch size of 16 videos. We add 50% dropout before the final classification layer.

Table C.1 shows our results. The accuracy-compute tradeoff is generally consistent with our results on EPIC-Kitchens. For example, with $r = 96$, we sacrifice 1.48% accuracy for a speedup of approximately 2×.

A threshold policy. We evaluate the ViTDet object detection model with a threshold policy. The threshold policy selects all tokens where the L2 norm of $\boldsymbol{e}$ exceeds a threshold h. We test $h = 0.2$, 1.0, and 5.0. See Table C.2 for results. The accuracy-compute tradeoff for the threshold policy is generally worse than for the top-r policy. This is likely due to the use of a constant threshold for all gates (we would ideally use a unique threshold for each gate).

C.3 Experiment Details

Fine-tuning ViTDet for VID. We initialize our model using COCO [139] pre-trained weights, and then trained on a combination of the ImageNet VID and ImageNet DET datasets, following common protocols in [38, 275]. We select images from the DET dataset that are of of the same 30 classes as in the

Table C.2: **A threshold policy.** Results for a threshold policy with the 1024-resolution ViTDet model. The policy selects tokens where the error $\boldsymbol{e}$ exceeds a threshold h.

Variant	h	mAP50 (%)	GFlops
Base model	—	82.93	467.4
Temporal	0.2	83.00	431.8
Temporal	1.0	82.75	294.1
Temporal	5.0	78.11	133.5

VID dataset. The training uses a batch size of 8, a maximum input resolution of 1024×1024, an initial learning rate of 10^{-4}, and a weight decay of 0.1. We use the AdamW optimizer [148] with linear warmup for a total of 5 epochs, with 10x learning rate decay from the 3rd epoch.

Fine-tuning the ViViT temporal model. We fine-tune the temporal submodel for 5 epochs. We use the AdamW optimizer [148] with a learning rate of 10^{-5}, weight decay of 0.05, and a batch size of 8 videos. We add 50% dropout before the final classification layer.

Arithmetic precision. We compute the product $A\boldsymbol{v}$ at half precision in the global self-attention operators of the Eventful model. Using half precision reduces the model's computational cost and memory footprint and has a negligible effect on accuracy. When evaluating runtimes, we also compute $A\boldsymbol{v}$ at half precision in the base model (this ensures a fair comparison).

Runtime experiments. For ViTDet, we evaluate CPU runtimes using one random video from VID (ID 00023010, containing 242 frames). On the GPU, we use 5 random videos. For ViViT, we evaluate CPU runtimes using 5 random videos from EPIC-Kitchens. On the GPU, we use 100 random videos. We use a consistent random seed across all experiment runs.

Operation counting. Our GFlop counts include the following types of operations: linear transforms, matrix multiplications, einsum operations (used in relative position embeddings), and additions. We count a multiply-accumulate

as a single operation. In Eventful Transformers, we additionally count operations required for updating the gate (additions and subtractions) and the extra additions in the sparse attention-value update. We only report operations in the Transformer backbones (e.g., we do not count anything in the object detection head).

D Stability and Robustness: Instant Video Models

D.1 Proofs

This section contains proofs of the oracle and collapse bounds from Section 5.4.

D.1.1 Oracle Bound

We assume that δ takes the form $\delta(\boldsymbol{a}, \boldsymbol{b}) = \zeta(\boldsymbol{a} - \boldsymbol{b})$ where ζ is a norm on $\mathbb{R}^d$. We define

$$u(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \sum_{t=1}^{\tau} \delta(\hat{\boldsymbol{y}}_t, \boldsymbol{y}_t) + \lambda \sum_{t=1}^{\tau-1} \delta(\hat{\boldsymbol{y}}_t, \hat{\boldsymbol{y}}_{t+1}) \tag{D.1}$$

$$= \sum_{t=1}^{\tau} \zeta(\hat{\boldsymbol{y}}_t - \boldsymbol{y}_t) + \lambda \sum_{t=1}^{\tau-1} \zeta(\hat{\boldsymbol{y}}_t - \hat{\boldsymbol{y}}_{t+1}) \tag{D.2}$$

as the bracketed expression in Equation 5.4, using the shorthand $\hat{\boldsymbol{y}}_t = f(\varepsilon_t(\boldsymbol{x}_t))$. Our objective is to show that for any $\hat{\boldsymbol{y}} \neq \boldsymbol{y}$, there exists some $\boldsymbol{y}^\dagger \neq \hat{\boldsymbol{y}}$ such that $u(\boldsymbol{y}^\dagger, \boldsymbol{y}) < u(\hat{\boldsymbol{y}}, \boldsymbol{y})$.

There are two possible cases: (1) $\hat{\boldsymbol{y}}_t \neq \boldsymbol{y}_t$ for at least one of the endpoints, meaning $\hat{\boldsymbol{y}}_1 \neq \boldsymbol{y}_1$ or $\hat{\boldsymbol{y}}_\tau \neq \boldsymbol{y}_\tau$; and (2) $\hat{\boldsymbol{y}}_t = \boldsymbol{y}_t$ at both endpoints.

We start with the first case. We assume that $\hat{\boldsymbol{y}}_1 \neq \boldsymbol{y}_1$, with the proof being

symmetric for $\hat{\boldsymbol{y}}_\tau \neq \boldsymbol{y}_\tau$. We propose $\boldsymbol{y}^\dagger$ where $\boldsymbol{y}_1^\dagger = \boldsymbol{y}_1$ and $\boldsymbol{y}_t^\dagger = \hat{\boldsymbol{y}}_t$ for $t > 1$. Starting with the definition of u,

$$u(\boldsymbol{y}^\dagger, \boldsymbol{y}) = \zeta(\boldsymbol{y}_1 - \boldsymbol{y}_1) + \lambda\zeta(\boldsymbol{y}_1 - \hat{\boldsymbol{y}}_2) + ... \tag{D.3}$$

$$= \lambda\zeta(\boldsymbol{y}_1 - \hat{\boldsymbol{y}}_2) + ... \tag{D.4}$$

$$u(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \zeta(\hat{\boldsymbol{y}}_1 - \boldsymbol{y}_1) + \lambda\zeta(\hat{\boldsymbol{y}}_1 - \hat{\boldsymbol{y}}_2) + ... \tag{D.5}$$

By the triangle inequality and absolute homogeneity of ζ,

$$u(\boldsymbol{y}^\dagger, \boldsymbol{y}) = \lambda\zeta(\boldsymbol{y}_1 - \hat{\boldsymbol{y}}_2) + ... \tag{D.6}$$

$$= \lambda\zeta((\boldsymbol{y}_1 - \hat{\boldsymbol{y}}_1) + (\hat{\boldsymbol{y}}_1 - \hat{\boldsymbol{y}}_2)) + ... \tag{D.7}$$

$$\leq \lambda\zeta(\boldsymbol{y}_1 - \hat{\boldsymbol{y}}_1) + \lambda\zeta(\hat{\boldsymbol{y}}_1 - \hat{\boldsymbol{y}}_2) + ... \tag{D.8}$$

$$= \lambda\zeta(\hat{\boldsymbol{y}}_1 - \boldsymbol{y}_1) + \lambda\zeta(\hat{\boldsymbol{y}}_1 - \hat{\boldsymbol{y}}_2) + ... \tag{D.9}$$

Assume that $\lambda < 1$. Then,

$$u(\boldsymbol{y}^\dagger, \boldsymbol{y}) \leq \lambda\zeta(\hat{\boldsymbol{y}}_1 - \boldsymbol{y}_1) + \lambda\zeta(\hat{\boldsymbol{y}}_1 - \hat{\boldsymbol{y}}_2) + ... \tag{D.10}$$

$$< \zeta(\hat{\boldsymbol{y}}_1 - \boldsymbol{y}_1) + \lambda\zeta(\hat{\boldsymbol{y}}_1 - \hat{\boldsymbol{y}}_2) + ... \tag{D.11}$$

$$= u(\hat{\boldsymbol{y}}, \boldsymbol{y}). \tag{D.12}$$

Therefore, $u(\boldsymbol{y}^\dagger, \boldsymbol{y}) < u(\hat{\boldsymbol{y}}, \boldsymbol{y})$.

To summarize, we have shown that if $\lambda < 1$, any $\hat{\boldsymbol{y}}$ where $\hat{\boldsymbol{y}}_1 \neq \boldsymbol{y}_1$ cannot minimize u. The same is true for $\hat{\boldsymbol{y}}_\tau \neq \boldsymbol{y}_\tau$.

Now we consider the second case, where $\hat{\boldsymbol{y}} \neq \boldsymbol{y}$ but their endpoints are the same. In this case, we must have $\hat{\boldsymbol{y}}_s \neq \boldsymbol{y}_s$ for some $1 < s < \tau$. We propose

$\boldsymbol{y}_s^\dagger = \boldsymbol{y}_s$ and $\boldsymbol{y}_t^\dagger = \hat{\boldsymbol{y}}_t$ for $t \neq s$. Starting again with the definition of u,

$$u(\boldsymbol{y}^\dagger, \boldsymbol{y}) = \cdots + \lambda\zeta(\hat{\boldsymbol{y}}_{t-1} - \boldsymbol{y}_t) + \zeta(\boldsymbol{y}_t - \boldsymbol{y}_t) + \lambda\zeta(\boldsymbol{y}_t - \hat{\boldsymbol{y}}_{t+1}) + \ldots \tag{D.13}$$

$$= \cdots + \lambda\zeta(\hat{\boldsymbol{y}}_{t-1} - \boldsymbol{y}_t) + \lambda\zeta(\boldsymbol{y}_t - \hat{\boldsymbol{y}}_{t+1}) + \ldots \tag{D.14}$$

$$u(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \cdots + \lambda\zeta(\hat{\boldsymbol{y}}_{t-1} - \hat{\boldsymbol{y}}_t) + \zeta(\hat{\boldsymbol{y}}_t - \boldsymbol{y}_t) + \lambda\zeta(\hat{\boldsymbol{y}}_t - \hat{\boldsymbol{y}}_{t+1}) + \ldots \tag{D.15}$$

By the triangle inequality and absolute homogeneity of ζ,

$$u(\boldsymbol{y}^\dagger, \boldsymbol{y}) = \cdots + \lambda\zeta(\hat{\boldsymbol{y}}_{t-1} - \boldsymbol{y}_t) + \lambda\zeta(\boldsymbol{y}_t - \hat{\boldsymbol{y}}_{t+1}) + \ldots \tag{D.16}$$

$$= \cdots + \lambda\zeta((\hat{\boldsymbol{y}}_{t-1} - \hat{\boldsymbol{y}}_t) + (\hat{\boldsymbol{y}}_t - \boldsymbol{y}_t)) + \lambda\zeta((\boldsymbol{y}_t - \hat{\boldsymbol{y}}_t) + (\hat{\boldsymbol{y}}_t - \hat{\boldsymbol{y}}_{t+1})) + \ldots \tag{D.17}$$

$$\leq \cdots + \lambda\zeta(\hat{\boldsymbol{y}}_{t-1} - \hat{\boldsymbol{y}}_t) + \lambda\zeta(\hat{\boldsymbol{y}}_t - \boldsymbol{y}_t) + \lambda\zeta(\boldsymbol{y}_t - \hat{\boldsymbol{y}}_t) + \lambda\zeta(\hat{\boldsymbol{y}}_t - \hat{\boldsymbol{y}}_{t+1}) + \ldots \tag{D.18}$$

$$= \cdots + \lambda\zeta(\hat{\boldsymbol{y}}_{t-1} - \hat{\boldsymbol{y}}_t) + 2\lambda\zeta(\hat{\boldsymbol{y}}_t - \boldsymbol{y}_t) + \lambda\zeta(\hat{\boldsymbol{y}}_t - \hat{\boldsymbol{y}}_{t+1}) + \ldots \tag{D.19}$$

Assume that $\lambda < 1/2$. Then,

$$u(\boldsymbol{y}^\dagger, \boldsymbol{y}) \leq \cdots + \lambda\zeta(\hat{\boldsymbol{y}}_{t-1} - \hat{\boldsymbol{y}}_t) + 2\lambda\zeta(\hat{\boldsymbol{y}}_t - \boldsymbol{y}_t) + \lambda\zeta(\hat{\boldsymbol{y}}_t - \hat{\boldsymbol{y}}_{t+1}) + \ldots \tag{D.20}$$

$$< \cdots + \lambda\zeta(\hat{\boldsymbol{y}}_{t-1} - \hat{\boldsymbol{y}}_t) + \zeta(\hat{\boldsymbol{y}}_t - \boldsymbol{y}_t) + \lambda\zeta(\hat{\boldsymbol{y}}_t - \hat{\boldsymbol{y}}_{t+1}) + \ldots \tag{D.21}$$

$$= u(\hat{\boldsymbol{y}}, \boldsymbol{y}) \tag{D.22}$$

Therefore, $u(\boldsymbol{y}^\dagger, \boldsymbol{y}) < u(\hat{\boldsymbol{y}}, \boldsymbol{y})$.

We have now shown that if $\lambda < 1/2$, any $\hat{\boldsymbol{y}}$ where $\hat{\boldsymbol{y}}_s \neq \boldsymbol{y}_s$ for $1 < s < \tau$ cannot minimize u. Combining this with the result from the first case, we have shown that for $\lambda < 1/2$, $\hat{\boldsymbol{y}} = \boldsymbol{y}$ is the unique global minimizer of u. □

D.1.2 Collapse Bound

To prove the collapse bound, we first prove the convexity of u. We then show that, if $\hat{\boldsymbol{y}}_1$ is fixed (cannot be modified by the stabilizer), $\hat{\boldsymbol{y}}_s = \hat{\boldsymbol{y}}_1$ for $1 < s \leq \tau$

is a local minimizer of u for $\lambda > \tau - 1$. Because u is convex, this makes $\hat{\boldsymbol{y}}_s = \hat{\boldsymbol{y}}_1$ the global minimizer.

Define $\boldsymbol{q}$ as the concatenation $(\hat{\boldsymbol{y}}, \boldsymbol{y})$, i.e., the vector input to u. Consider two such inputs $\boldsymbol{q}$ and $\boldsymbol{q}^\dagger$. u satisfies the triangle inequality;

$$u(\boldsymbol{q} + \boldsymbol{q}^\dagger) = \sum_{t=1}^{\tau} \zeta(\hat{\boldsymbol{y}}_t - \boldsymbol{y}_t + \hat{\boldsymbol{y}}_t^\dagger - \boldsymbol{y}_t^\dagger) + \lambda \sum_{t=1}^{\tau-1} \zeta(\hat{\boldsymbol{y}}_t - \hat{\boldsymbol{y}}_{t+1} + \hat{\boldsymbol{y}}_t^\dagger - \hat{\boldsymbol{y}}_{t+1}^\dagger) \tag{D.23}$$

$$\leq \sum_{t=1}^{\tau} \zeta(\hat{\boldsymbol{y}}_t - \boldsymbol{y}_t) + \lambda \sum_{t=1}^{\tau-1} \zeta(\hat{\boldsymbol{y}}_t - \hat{\boldsymbol{y}}_{t+1}) + \sum_{t=1}^{\tau} \zeta(\hat{\boldsymbol{y}}_t^\dagger - \boldsymbol{y}_t^\dagger) + \lambda \sum_{t=1}^{\tau-1} \zeta(\hat{\boldsymbol{y}}_t^\dagger - \hat{\boldsymbol{y}}_{t+1}^\dagger) \tag{D.24}$$

$$= u(\boldsymbol{q}) + u(\boldsymbol{q}^\dagger), \tag{D.25}$$

and is absolutely homogeneous;

$$u(c\boldsymbol{q}) = \sum_{t=1}^{\tau} \zeta(c\hat{\boldsymbol{y}}_t - c\boldsymbol{y}_t) + \lambda \sum_{t=1}^{\tau-1} \zeta(c\hat{\boldsymbol{y}}_t - c\hat{\boldsymbol{y}}_{t+1}) \tag{D.26}$$

$$= |c| \sum_{t=1}^{\tau} \zeta(\hat{\boldsymbol{y}}_t - \boldsymbol{y}_t) + |c|\lambda \sum_{t=1}^{\tau-1} \zeta(\hat{\boldsymbol{y}}_t - \hat{\boldsymbol{y}}_{t+1}) \tag{D.27}$$

$$= |c| u(\boldsymbol{q}), \tag{D.28}$$

implying u is convex;

$$u(r\boldsymbol{q} + (1-r)\boldsymbol{q}^\dagger) \leq u(r\boldsymbol{q}) + u((1-r)\boldsymbol{q}^\dagger) \tag{D.29}$$

$$= |r| u(\boldsymbol{q}) + |1-r| u(\boldsymbol{q}^\dagger) \tag{D.30}$$

$$= r u(\boldsymbol{q}) + (1-r) u(\boldsymbol{q}^\dagger). \tag{D.31}$$

where $0 \leq r \leq 1$. Because u is jointly convex on $\boldsymbol{q}$, it is also convex with respect to $(\hat{\boldsymbol{y}}_2, \ldots, \hat{\boldsymbol{y}}_\tau)$ when $\boldsymbol{y}$ and $\hat{\boldsymbol{y}}_1$ are fixed.

Our goal is now to show that, assuming $\boldsymbol{y}$ and $\hat{\boldsymbol{y}}_1$ are fixed, $\hat{\boldsymbol{y}}_s = \hat{\boldsymbol{y}}_1$ for $1 < s \leq \tau$ is a local minimizer of u. That is, we want to find the λ regime where any small movement away from this point increases the value of u. Assume a perturbed prediction

$$\boldsymbol{y}^\dagger = (\hat{\boldsymbol{y}}_1 + \boldsymbol{p}_1, \hat{\boldsymbol{y}}_1 + \boldsymbol{p}_2, \dots, \hat{\boldsymbol{y}}_1 + \boldsymbol{p}_\tau), \tag{D.32}$$

where $\boldsymbol{p}_1 = 0$. Starting with the definition of u,

$$u(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \sum_{t=1}^{\tau} \zeta(\hat{\boldsymbol{y}}_1 - \boldsymbol{y}_t) + \lambda \sum_{t=1}^{\tau-1} \zeta(\hat{\boldsymbol{y}}_1 - \hat{\boldsymbol{y}}_1) \tag{D.33}$$

$$= \sum_{t=1}^{\tau} \zeta(\hat{\boldsymbol{y}}_1 - \boldsymbol{y}_t) \tag{D.34}$$

$$u(\boldsymbol{y}^\dagger, \boldsymbol{y}) = \sum_{t=1}^{\tau} \zeta(\hat{\boldsymbol{y}}_1 + \boldsymbol{p}_t - \boldsymbol{y}_t) + \lambda \sum_{t=1}^{\tau-1} \zeta((\hat{\boldsymbol{y}}_1 + \boldsymbol{p}_t) - (\hat{\boldsymbol{y}}_1 + \boldsymbol{p}_{t+1})) \tag{D.35}$$

$$= \sum_{t=1}^{\tau} \zeta(\hat{\boldsymbol{y}}_1 + \boldsymbol{p}_t - \boldsymbol{y}_t) + \lambda \sum_{t=1}^{\tau-1} \zeta(\boldsymbol{p}_t - \boldsymbol{p}_{t+1}) \tag{D.36}$$

Let

$$\theta = \operatorname*{argmax}_{t \in 1:\tau} \zeta(\boldsymbol{p}_t) \tag{D.37}$$

be the time step with the largest-magnitude perturbation, and let $\phi = \zeta(\boldsymbol{p}_\theta)$ be the magnitude of this perturbation. Applying the reverse triangle inequality

to the first summation in Equation D.36 (the accuracy term), we have

$$\sum_{t=1}^{\tau} \zeta(\hat{\boldsymbol{y}}_1 + \boldsymbol{p}_t - \boldsymbol{y}_t) = \sum_{t=1}^{\tau} \zeta((\hat{\boldsymbol{y}}_1 - \boldsymbol{y}_t) - (-\boldsymbol{p}_t)) \tag{D.38}$$

$$\geq \sum_{t=1}^{\tau} [\zeta(\hat{\boldsymbol{y}}_1 - \boldsymbol{y}_t) - \zeta(-\boldsymbol{p}_t)] \tag{D.39}$$

$$= \sum_{t=1}^{\tau} \zeta(\hat{\boldsymbol{y}}_1 - \boldsymbol{y}_t) - \sum_{t=1}^{\tau} \zeta(\boldsymbol{p}_t) \tag{D.40}$$

$$= \sum_{t=1}^{\tau} \zeta(\hat{\boldsymbol{y}}_1 - \boldsymbol{y}_t) - \sum_{t=2}^{\tau} \zeta(\boldsymbol{p}_t) \tag{D.41}$$

$$\geq \sum_{t=1}^{\tau} \zeta(\hat{\boldsymbol{y}}_1 - \boldsymbol{y}_t) - (\tau - 1)\phi \tag{D.42}$$

So, introducing the perturbation reduces the accuracy term by at most $(\tau - 1)\phi$. Now considering the second summation (the stability term),

$$\lambda \sum_{t=1}^{\tau-1} \zeta(\boldsymbol{p}_t - \boldsymbol{p}_{t+1}) \geq \lambda \sum_{t=1}^{\theta} \zeta(\boldsymbol{p}_t - \boldsymbol{p}_{t+1}) \tag{D.43}$$

$$\geq \lambda \zeta(\boldsymbol{p}_1 - \boldsymbol{p}_\theta) \tag{D.44}$$

$$= \lambda \zeta(\boldsymbol{p}_\theta) \tag{D.45}$$

$$= \lambda \phi. \tag{D.46}$$

So, introducing the perturbation increases the stability term by at least $\lambda\phi$.

Therefore, if $\lambda > \tau - 1$, the overall change in u is positive, and we have shown that $\hat{\boldsymbol{y}}_s = \hat{\boldsymbol{y}}_1$ for $1 < s \leq \tau$ is a local minimizer of u. By convexity of u, this point is also a global minimizer. □

D.2 Transport Metric

In this section, we propose an alternate metric for use with our unified loss. Specifically, we describe a variant of the Wasserstein metric that accounts for the spatial structure of an image or feature tensor.

Let $\boldsymbol{z}_1, \boldsymbol{z}_2 \in \mathbb{R}^{h \times w}$ be two image or feature map channels, and let $\boldsymbol{a} = \boldsymbol{z}_1 - \boldsymbol{z}_2$. We define the transport distance $\mathcal{T}(\boldsymbol{a}) = \zeta(\boldsymbol{a}) = \delta(\boldsymbol{z}_1, \boldsymbol{z}_2)$ as the minimum cost of a linear optimization. Intuitively, this optimization finds the shortest correspondence from $\boldsymbol{m}$ to zero. The correspondence can employ three mechanisms: (1) mass movement from a positive region to a negative region, with cost proportional to mass and distance, (2) mass destruction, with cost proportional to mass, and (3) mass creation, also with cost proportional to mass. The cost to create or destroy a unit of mass is γ.

Formally, we solve the following optimization:

$$\mathcal{T}(\boldsymbol{a}) = \min_{\boldsymbol{m},\boldsymbol{p},\boldsymbol{c}} \left[\sum_{i,j} d_{ij} m_{ij} + \gamma \sum_{i} (p_i + c_i) \right] \tag{D.47}$$

subject to

$$a_i + p_i - c_i - \sum_j m_{ij} = 0 \quad \forall i \tag{D.48}$$

$$p_i > 0 \quad \forall i \tag{D.49}$$

$$c_i > 0 \quad \forall i \tag{D.50}$$

$$m_{ij} > 0 \quad \forall i, j \tag{D.51}$$

where d_{ij} is the distance (Euclidean) from pixel i to pixel j, m_{ij} is the mass moved from pixel i to pixel j, and p_i and c_i are the mass production and consumption at pixel i. Both i and j are in the range $1 \ldots h \times w$.

At first glance, this problem may seem computationally infeasible because

the number of d_{ij} parameters equals the number of pixels squared, e.g., a trillion parameters for a one-megapixel image. Fortunately, we can reduce the number of parameters to $\sim (h \times w)$ by pruning all edges where $d_{ij} > 2\gamma$. Intuitively, if $d_{ij} > 2\gamma$, the cost to destroy a unit of mass at location i and recreate it at location j is less than the cost to move it from i to j.

Despite pruning, this optimization remains somewhat impractical. Finding a solution takes ∼seconds using the open-source solvers available in SciPy. These runtimes make loss evaluation the primary bottleneck during training. We might be able to reduce runtime using other solvers—either general-purpose commercial solvers or specialized optimal transport solvers. We leave this as future work.

D.3 Composing Stabilizers

Often, a deployed model will be faced with several simultaneous corruptions. One option in this scenario is to train a single stabilizer on the expected combination of corruptions. However, doing so requires full knowledge of the corruptions at training time. In this section, we explore an alternate approach: composing (fusing) single-purpose stabilizers without additional training.

We consider controlled stabilizers without spatial fusion. Assume we have two single-corruption stabilizers, denoted by superscripts 1 and 2. The output $\tilde{\boldsymbol{z}}_{i,t}$ of the fused stabilizers at layer i is

$$\tilde{\boldsymbol{z}}_{i,t} = \boldsymbol{\beta}^2_{i,t} \odot \tilde{\boldsymbol{z}}^1_{i,t} + (1 - \boldsymbol{\beta}^2_{i,t}) \odot \tilde{\boldsymbol{z}}_{i,t-1}, \tag{D.52}$$

$$\tilde{\boldsymbol{z}}^1_{i,t} = \boldsymbol{\beta}^1_{i,t} \odot \boldsymbol{z}_{i,t} + (1 - \boldsymbol{\beta}^1_{i,t}) \odot \tilde{\boldsymbol{z}}^1_{i,t-1}, \tag{D.53}$$

$$\boldsymbol{\beta}^1_{i,t} = \sigma(h^1_i(g^1(\boldsymbol{x}_t, \boldsymbol{x}_{t-1}))), \tag{D.54}$$

$$\boldsymbol{\beta}^2_{i,t} = \sigma(h^2_i(g^2(\boldsymbol{x}_t, \boldsymbol{x}_{t-1}))). \tag{D.55}$$

Note that we have removed the feature-space inputs $\boldsymbol{z}_t$, $\tilde{\boldsymbol{z}}_{t-1}$, and $\boldsymbol{z}_{t-1}$ to the

stabilization heads. In experiments, we found this was necessary to prevent unintended interactions between the stabilizers.

The above formulation is roughly equivalent to applying a single stabilizer with decay $\boldsymbol{\beta}^1_{i,t} \odot \boldsymbol{\beta}^2_{i,t}$. We can modify the method slightly to make this strictly true, thereby obtaining a commutative composition. We replace $\tilde{\boldsymbol{z}}^1_{i,t-1}$ in Equation D.53 with $\tilde{\boldsymbol{z}}_{i,t-1}$, obtaining

$$\tilde{\boldsymbol{z}}_{i,t} = \boldsymbol{\beta}^2_{i,t} \odot (\boldsymbol{\beta}^1_{i,t} \odot \boldsymbol{z}_{i,t} + (1 - \boldsymbol{\beta}^1_{i,t}) \odot \tilde{\boldsymbol{z}}_{i,t-1}) + (1 - \boldsymbol{\beta}^2_{i,t}) \odot \tilde{\boldsymbol{z}}_{i,t-1}, \tag{D.56}$$

$$= \boldsymbol{\beta}^2_{i,t} \odot \boldsymbol{\beta}^1_{i,t} \odot \boldsymbol{z}_{i,t} + (\boldsymbol{\beta}^2_{i,t} - \boldsymbol{\beta}^2_{i,t} \odot \boldsymbol{\beta}^1_{i,t} + 1 - \boldsymbol{\beta}^2_{i,t}) \odot \tilde{\boldsymbol{z}}_{i,t-1}, \tag{D.57}$$

$$= \boldsymbol{\beta}^2_{i,t} \odot \boldsymbol{\beta}^1_{i,t} \odot \boldsymbol{z}_{i,t} + (1 - \boldsymbol{\beta}^2_{i,t} \odot \boldsymbol{\beta}^1_{i,t}) \odot \tilde{\boldsymbol{z}}_{i,t-1}. \tag{D.58}$$

Intuitively, the fused stabilizer retains the current features $\boldsymbol{z}_{t,i}$ only if both decays are near one; this indicates that neither controller backbone detected corruption-induced instability.

In our experiments, we use the initial non-commutative version (it was slightly easier to implement). However, we expect the two formulations to give similar results in general.

D.4 Method Details

This section provides further details of our method and implementation, including the particular architecture we use for the controller, our approach for training initialization, and a formal definition of the spatial fusion mechanism.

D.4.1 Controller Architecture

The controller backbone uses a simple convolutional architecture with 7 layers. The backbone has 32 channels for HDRNet and NAFNet, and 16 channels for Depth Anything. Controller heads have 4 layers. The number of channels in the last head layer depends on the shape of $\boldsymbol{z}$ and the size of the fusion

kernel. For other head layers, we use 64 channels for HDRNet and NAFNet, and 32 channels for Depth Anything. All convolutions use a 3×3 kernel, and all except the head outputs employ a leaky ReLU with negative slope 0.01.

D.4.2 Initialization

When training, we initialize such that the predicted β values are near 1. This corresponds to a rapid decay of the past state and less stabilization; i.e., the stabilizers are initialized close to an identity. For controlled stabilizers, this can be achieved by adding a final bias to h (before the sigmoid) and initializing this bias to a sufficiently positive value v. For the simple learned stabilizer (without a controller), the trained parameters are logit values l used to generate a decay $\in [0, 1]$ via $\beta = \sigma(l)$. Again, we initialize these logits to a positive value v. For both the simple learned and controlled variants, we find that $v = 4$ (resulting in $\beta \approx 0.98$) works well.

D.4.3 Spatial Fusion

Let $\mathcal{N}$ be a spatial neighborhood around the pixel to be stabilized. Let c denote the channel index, j the index of the stabilized pixel, $k \in \mathcal{N}$ the neighbor pixel index, and $l \in 1 : m$ an index into the kernel $\boldsymbol{\eta}$. The output $\tilde{z}_{t,c,j}$ of the spatial fusion stabilizer is given by

$$\tilde{z}_{t,c,k} = \eta_{t,c,m+1} z_{t,c,j} + \sum_{(k,l)\in(\mathcal{N},1:m)} \eta_{t,c,l} \tilde{z}_{t,c,k} \tag{D.59}$$

$$\boldsymbol{\eta}_{t,c} = \mathrm{Softmax}([h(g(\boldsymbol{x}_t, \boldsymbol{x}_{t-1}), \boldsymbol{z}_t, \boldsymbol{z}_t - 1)]_{cm:cm+m-1}, 0), \tag{D.60}$$

The summation iterates over locations in the neighborhood, with the kernel index l always corresponding to the neighbor index k. The indexing on the output of h extracts the m channels needed to construct the kernel $\boldsymbol{\eta}_{t,c}$. Note that we have dropped the layer index i here for brevity.

D.5 Experiment Details

This section contains experiment details and hyperparameter values.

D.5.1 Image Enhancement

Base model fine-tuning. We train for 80 epochs (2k iterations per epoch), using the Adam optimizer [113], an MSE loss, and batches of 8 randomly sampled frames. The learning rate is initially set to 10^{-4} and is scaled by 0.1 after epochs 40 and 60.

Stabilizer training and evaluation. Each training batch consists of one randomly sampled video snippet containing $\tau = 8$ consecutive frames. Gradients are computed using BPTT. Stabilizers are trained for 20 epochs (4k iterations per epoch) using the Adam optimizer and our unified loss with $\delta = ||\cdot||_2$. The learning rate is initialized to 10^{-3} for the simple learned stabilizer and 10^{-4} for other variants, and is reduced by a factor of 10 after epochs 10 and 15. We evaluate stabilizers on the validation set, processing each video in a single pass per video (i.e., $\tau \gg 8$ at evaluation).

D.5.2 Video Denoising

Base model fine-tuning. Each batch consists of 8 randomly sampled, randomly cropped patches of size 256×256. We train for 20 epochs (2k steps per epoch) with Adam and an MSE loss. We set the initial learning rate to 10^{-4}, scaling by 0.1 after epochs 10 and 15.

Stabilizer training and evaluation. Each batch consists of one randomly sampled, randomly cropped video snippet of size 256×256 containing $\tau = 8$ consecutive frames. We train for 20 epochs (2k steps per epoch) using Adam and the unified loss with $\delta = ||\cdot||_2$. The initial learning rate is set to 10^{-2} for the simple learned stabilizer and 10^{-4} for the controlled and spatial variants. In

Figure D.1: **VisionSim sequences.** Showing scenes bachelors-quarters, classroom, library-homeoffice, and restaurant. For each, we show frames 10, 110, 210, 310, 410, and 510.

all cases, we scale the learning rate by 0.1 after epochs 10 and 15. We evaluate stabilizers on the validation set in a single pass per video ($\tau \gg 8$).

D.5.3 Corruption Robustness

Depth training dataset. We use the VisionSim [101] framework to generate a dataset with ground-truth depth labels, according to the instructions provided in the large-scale datasets tutorial. The resulting dataset contains 50 scenes with 59950 total frames and resolution 800×800. All scenes are indoor and exclusively contain ego-motion. See Figure D.1 for several examples.

Depth metrics. Depth Anything predicts relative disparity (inverse depth),

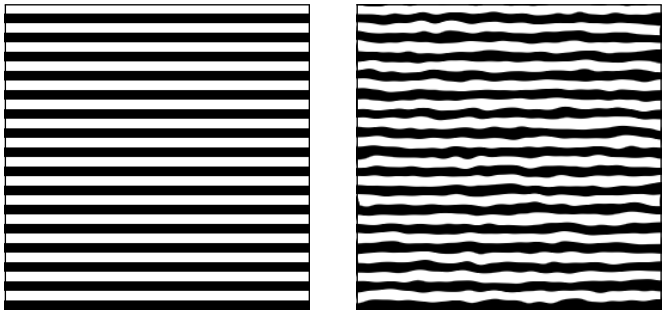


Figure D.2: **Elastic transform.** A dummy image before and after applying the elastic transform, using the same parameters as our experiments (magnitude $\alpha = 50.0$ and smoothness $\sigma = 5.0$). The image size is 256×256, and the lines have width 16.

which requires an affine alignment to the ground truth before computing metrics (see [184] for details). After this alignment, we evaluate the standard AbsRel and Delta–1 ($\delta > 1.25$) metrics. We exclude outliers by clipping the aligned depth to a maximum of 200. It is critical to measure instability *after* alignment; otherwise, the network can achieve arbitrarily low instability without harming AbsRel and Delta–1 by scaling predictions to a small range around zero.

Depth stabilizer training. We train Depth Anything stabilizers on randomly sampled snippets of length $\tau = 8$, randomly cropped to size 512×512. We train using Adam for 20 epochs (4k iterations per epoch) with a batch size of one. The learning rate is initialized to 10^{-4} and reduced by 0.1 after epochs 10 and 15. We evaluate the unified loss ($\delta = ||\cdot||_2$) after affine alignment.

Image corruptions. We generate elastic distortions using the Torchvision ElasticTransform class with default settings (magnitude $\alpha = 50.0$ and smoothness $\sigma = 5.0$). See Figure D.2 for a visualization of the elastic transform.

For image denoising, we add all corruptions *after* Gaussian noise. For example, dropped frames contain zeros, not zero-mean Gaussian noise.

D.5.4 Adverse Weather Robustness

Base model fine-tuning. We use the same schedule and hyperparameters as in Section D.5.2. Each epoch consists of 1k training steps.

Stabilizer training and evaluation. We again use the same schedule and hyperparameters as in Section D.5.2. The initial learning rate is set to 10^{-4}, and each epoch consists of 800 steps.

D.5.5 Compute Requirements

We train and evaluate on a compute cluster largely using RTX A4500 GPUs. Fine-tuning and stabilizer training take 1–2 days on a single GPU, and full evaluation takes 1–3 hours.

D.6 Additional Results

D.6.1 Stabilizer Composition

We evaluate composed stabilizers (Appendix D.3) on the NFS denoising task ($\sigma = 0.1$) using the NAFNet model. We train single-purpose corruption stabilizers using the same hyperparameters as other experiments, then evaluate composition under all possible two-corruption pairings. Results are shown in Table D.1. Generally, stabilizer composition is effective if the second corruption does not significantly change the appearance of the first. For example, JPEG → impulse works better than impulse → JPEG. JPEG compression obscures high-frequency impulse noise, thereby interfering with the impulse noise controller backbone. This limitation could likely be addressed through data augmentation during stabilizer training, which would make the backbones more robust to changes in corruption appearance.

		Ours (stabilized)			Base model (unstabilized)		
First corruption	Second corruption	PSNR	SSIM	Instability	PSNR	SSIM	Instability
Patch drop	Elastic distortion	29.53	0.838	22.64	18.56	0.660	152.40
Patch drop	Frame drop	33.88	0.919	19.15	17.47	0.658	218.69
Patch drop	JPEG artifacts	29.07	0.845	31.67	18.51	0.637	152.60
Patch drop	Impulse noise	24.83	0.769	60.89	17.49	0.367	153.96
Elastic distortion	Patch drop	29.93	0.850	19.24	18.42	0.657	155.24
Elastic distortion	Frame drop	28.01	0.780	17.79	25.65	0.769	129.61
Elastic distortion	JPEG artifacts	29.11	0.842	18.46	25.94	0.759	53.75
Elastic distortion	Impulse noise	27.20	0.785	16.22	22.95	0.480	69.70
Frame drop	Patch drop	33.84	0.916	19.79	17.50	0.659	218.18
Frame drop	Elastic distortion	29.67	0.844	18.69	25.65	0.768	130.22
Frame drop	JPEG artifacts	25.99	0.737	17.59	26.56	0.720	124.21
Frame drop	Impulse noise	28.95	0.807	57.28	23.14	0.467	103.00
JPEG artifacts	Patch drop	30.55	0.857	21.88	18.41	0.638	154.20
JPEG artifacts	Elastic distortion	29.19	0.844	18.16	25.81	0.755	54.48
JPEG artifacts	Frame drop	25.87	0.737	17.16	26.56	0.720	124.21
JPEG artifacts	Impulse noise	28.40	0.810	19.74	23.93	0.475	65.24
Impulse noise	Patch drop	30.14	0.831	24.61	16.98	0.351	163.34
Impulse noise	Elastic distortion	26.73	0.790	17.86	23.38	0.600	62.01
Impulse noise	Frame drop	28.94	0.794	21.21	22.66	0.460	138.58
Impulse noise	JPEG artifacts	25.25	0.599	39.47	21.55	0.352	89.25

Table D.1: **Stabilizer composition.** The effectiveness of composing stabilizers for different combinations and orderings of input corruptions. Results are for NAFNet on the NFS dataset with moderate noise ($\sigma = 0.1$). The base model evaluated without corruptions achieves PSNR 36.6, SSIM 0.945, and instability 26.2.

D.6.2 Semantic Segmentation

Task, dataset, and base model. In this subsection, we analyze a higher-level prediction task: semantic segmentation. We use the DeepLabv3+ model [34]—specifically, the MobileNet variant published by [55]. We train and evaluate on the VIPER dataset [192], which contains video sequences captured in a game engine (GTA V) and automatically labeled for various vision tasks. The predefined training and validation splits contain 77 sequences (134097 frames) and 47 sequences (49815 frames), respectively. We measure prediction quality using pixel accuracy and mIoU on the predefined validation set. Due to the discrete nature of predictions, we report categorical instability—the fraction of pixels whose category changes between frames (equivalent to $||\cdot||_0$).

Experiment protocol. We fine-tune the unstabilized model, starting with the Cityscapes [43] weights published by [55]. A 1×1 convolution is applied to the output logits to adapt the number of classes for VIPER. We fine-tune for 60 epochs (1925 batches per epoch), using a batch size of 16 and a cross-entropy loss. Adam was used with an initial learning rate of 10^{-4}, scaled by 0.1 after epochs 20 and 40.

After fine-tuning, stabilizers are attached to the model input, the model output, each InvertedResidual layer, the "aspp" block, the "project" block, and the "classifier" block. We then freeze the fine-tuned weights and train stabilizers on snippets of length $\tau = 8$ with $\lambda = 0.4$. Here we tried both $||\cdot||_2$ and cross-entropy for δ. Cross-entropy gave slightly better results, despite not satisfying the formal criteria in Section 5.4 (e.g., it is not symmetric). Therefore, we report results with cross-entropy in the remaining experiments.

When training stabilizers, each batch consists of one snippet of length $\tau = 8$ frames. We train for 60 epochs (3080 steps per epoch), using Adam with the same learning rate schedule as in fine-tuning.

Results. The unstabilized model achieves categorical instability 0.079, mIoU

Corruption	Method	mIoU	Accuracy	Instability
Patch drop	Base model	0.060	0.164	0.454
	Ours	0.405	0.896	0.064
Elastic distortion	Base model	0.377	0.888	0.090
	Ours	0.403	0.898	0.064
Frame drop	Base model	0.369	0.829	0.209
	Ours	0.406	0.898	0.063
JPEG artifacts	Base model	0.109	0.313	0.216
	Ours	0.337	0.862	0.066
Impulse noise	Base model	0.051	0.300	0.312
	Ours	0.399	0.894	0.065

Table D.2: **Segmentation robustness.** For all corruptions, our method simultaneously improves mIoU, pixel accuracy, and instability. The improvement is largest for corruptions that significantly change the appearance of the input, e.g., impulse noise.

0.406, and pixel accuracy 0.900. After adding stabilizers, we obtain instability 0.059, mIoU 0.411, and pixel accuracy 0.901. As for other tasks, there is a marked improvement in stability, along with an increase in accuracy (mIoU).

D.6.3 Segmentation Robustness

Task, dataset, and base model. In this subsection, we evaluate the segmentation model DeepLabv3+ against the five corruptions from Section 5.6.3 (patch drop, elastic distortion, frame drop, JPEG artifacts, and impulse noise). We use the same dataset (VIPER) and metrics as in Section D.6.2.

Experiment protocol. We follow the same protocol as in Section D.6.2 when training and evaluating corruption stabilizers.

Results. See Table D.6.3 for metric values, and Figure D.8 for sample predictions. Our method improves both task metrics and stability across all corruptions. For patch drop, JPEG artifacts, and impulse noise, adding stabilizers allows the model to recover from catastrophic prediction failures.

D.6.4 DAVIS Denoising

Task, dataset, and base model. In addition to NFS, we evaluate NAFNet denoising on the standard DAVIS benchmark [134, 181, 220]. DAVIS contains 50 videos (3455 frames) collected at 24 FPS. We use the dataset's predefined train/validation split and scale images to a short edge length of 480 [220]. Following from prior work [134, 220], we evaluate with a noise level of $40/255 \approx 0.16$.

Experiment protocol. We fine-tune the base model and train stabilizers following the same procedure as for NFS (see Sections 5.6.2 and D.5.2). Fine-tuning epochs contain 3k steps, and stabilizer training epochs have 2.4k steps.

Results. See Figure D.3 and Table D.5 for results. Overall, the behavior is similar to the NFS results in Section 5.6.2. However, the "win-win" region (where both accuracy and stability are improved) is smaller for DAVIS. This is likely caused by DAVIS's 10× lower frame rate, which corresponds to higher inter-frame motion and lower frame-to-frame correlation.

D.6.5 Adversarial Robustness

In addition to natural corruptions, we evaluate our method in the presence of adversarial corruptions. We consider a setting where the attacker has knowledge of the base model and its parameters, but not of the stabilizers. We reason that because the stabilizers have fewer parameters than the base model and can be trained more quickly, a defender could update the stabilizer parameters after a weight leak rather than retrain the entire model.

Task and model. We evaluate adversarial robustness on a binary classification task derived from the DAVIS dataset. Frames are processed by tightly cropping around an object's segmentation mask, treating individual instances as separate images, and labeling each crop as human or nonhuman. As our

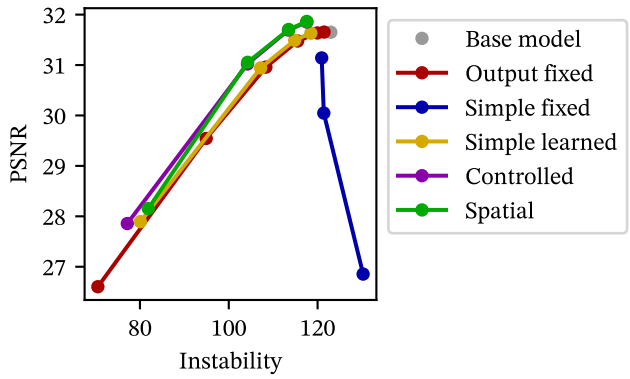


Figure D.3: **DAVIS denoising.** We obtain the best quality/stability tradeoff when using a stabilization controller ("controlled" and "spatial"). On DAVIS, spatial fusion does not offer a significant advantage compared to a basic controlled stabilizer. DAVIS has relatively high inter-frame motion, which often exceeds the size of the spatial kernel (i.e., the maximum translation achievable with spatial fusion).

backbone, we use ResNet-50 [80] pre-trained on ImageNet. We replace the original 1000-class output layer with a two-class linear layer.

Experiment protocol. We freeze all pre-trained weights except those in the final residual block ("layer4") and the new classification head. We fine-tune these parameters for 100 epochs on the binary classification task. We start with a learning rate of 10^{-2}, decreasing it by a factor of 0.1 at epochs 40 and 80, and employ frame-level data augmentation consisting of random rotation, horizontal flip, and color jitter. Each training sample consists of a sequence of eight consecutive frames.

Then, we generate adversarial examples using the iterative Fast Gradient Sign Method (I-FGSM) with an overall perturbation bound $\varepsilon = 0.1$ and 20 iterations per image [66]. As the attacker does not know the ground-truth class, for each image, we randomly apply either the sign of the gradient step or its negation. We selected these hyperparameters because they produce a substantial drop in baseline accuracy, creating a clear opportunity for the

stabilizers to improve performance. Then, to defend against the attack, we add controlled stabilizers to each bottleneck block of the ResNet model. All original ResNet parameters are frozen, and only stabilizers are trained for 20 epochs under the same I-FGSM attack settings as above. The stabilizer training uses an initial learning rate of 10^{-4}, reduced by a factor of 0.1 after epochs 1 and 10, and each batch consists of two sequences of eight frames to ensure well-defined gradients. No additional augmentations are applied during this phase.

Results. Under our adversarial setup, the ResNet-50 fine-tuned baseline achieves 77.0% accuracy, while our stabilizer-augmented model reaches 88.8%, an absolute improvement of 11.8%. These results demonstrate that the proposed stabilizer modules can significantly improve resilience to adversarial attacks without requiring complete retraining.

D.6.6 Spatial Fusion Failures

Discussion. In the denoising experiments (Section 5.6.2), we observed a significant PSNR reduction when using the spatial fusion method under extreme noise. This reduction only appears when evaluating on long sequences.

A closer examination of the outputs reveals blurring/ghosting artifacts that appear after some time has passed. These artifacts look like hard edges "bleeding out" into the surrounding regions. We believe these failures are related to the spatial fusion stabilizer on the output layer. Without this stabilizer, the network output is biased toward the current input frame (due to the network architecture, which predicts a noise residual). Adding a spatial fusion stabilizer to the output provides an independent mechanism for information to flow between pixels, thereby weakening this bias.

Experiment protocol. We ran an experiment to determine whether spatial fusion failures can be mitigated by training on longer sequences. We trained the spatial fusion stabilizer under extreme noise on sequences of length $\tau = 8$

(the default in our other experiments) and $\tau = 16$. We reduced the training patch size from 256×256 to 180×180 to compensate for increased training memory requirements on longer sequences. For $\tau = 8$, we doubled the number of iterations per epoch due to the lower number of frames in each iteration. We evaluated on the full validation set containing long sequences.

Results. Training with $\tau = 8$ gave validation PSNR 22.70 and instability 11.04, whereas training with $\tau = 16$ gave PSNR 23.80 and instability 10.16. We expect this trend of improvement to hold as we further increase the training sequence length.

D.6.7 Uncertainty Estimate

Experiment protocol. To estimate uncertainty in our results, we train and evaluate the spatial fusion stabilizer eight times with different random seeds. We consider image enhancement (HDRNet) for the moderate-strength local Laplacian operator ($\alpha = 0.25$). The random seed determines the stabilizer weight initialization and the training data shuffle. The training and evaluation protocol is identical to that in Sections 5.6.1 and D.5.1.

Results. PSNR ranges from a minimum of 32.17 to a maximum of 32.32, with a mean of 32.25. SSIM has range 0.926–0.929 (mean 0.927), and instability has range 28.61–28.80 (mean 28.71). For all metrics, variations around the mean are $< 0.5\%$.

D.6.8 Figures

Figure D.4 shows example sequences denoised under extreme Gaussian noise ($\sigma = 0.6$). We highlight regions of the image where instability is especially prominent. Differences are more noticeable in videos; we encourage the reader to view the video files included with the supplementary material.

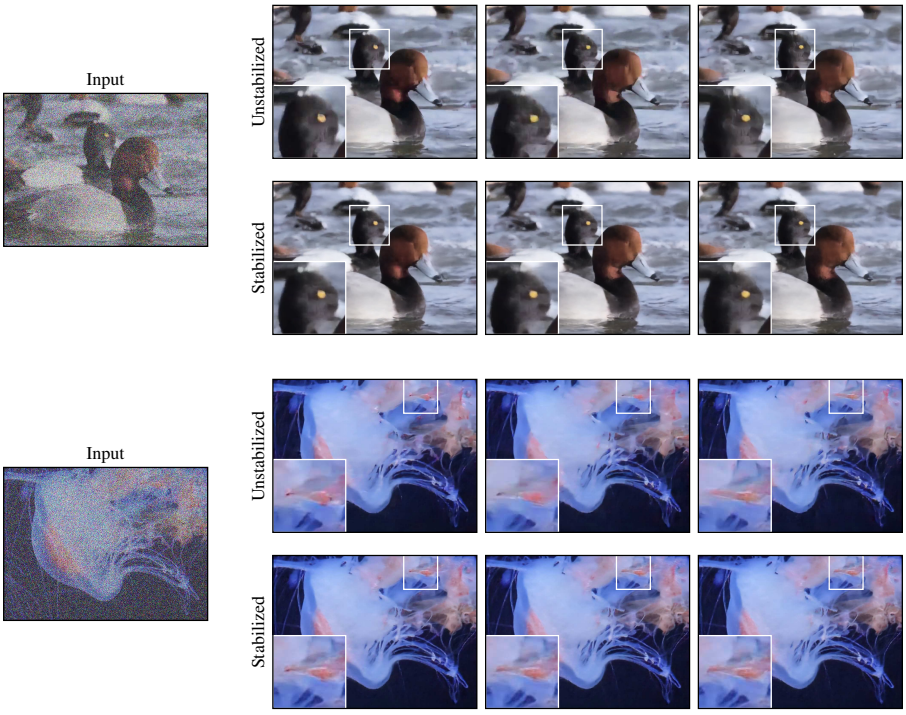


Figure D.4: **Denoising under extreme noise.** As we increase the level of image noise, frame-wise temporal inconsistency becomes more severe, to the point that it becomes apparent when comparing static images. In the top sequence, we see shifting texture in the duck's head in the unstabilized features. In the jellyfish sequence, the denoiser hallucinates inconsistent spatial structure between frames. In both cases, adding a stabilizer noticeably improves temporal consistency. We encourage the reader to view the corresponding video files included with the supplement.

Figures D.5, D.6, D.7, and D.8 contain examples of corruption robustness for image enhancement, denoising, depth estimation, and segmentation, respectively. Figure D.9 illustrates improved weather robustness for denoising on RobustSpring D.9.

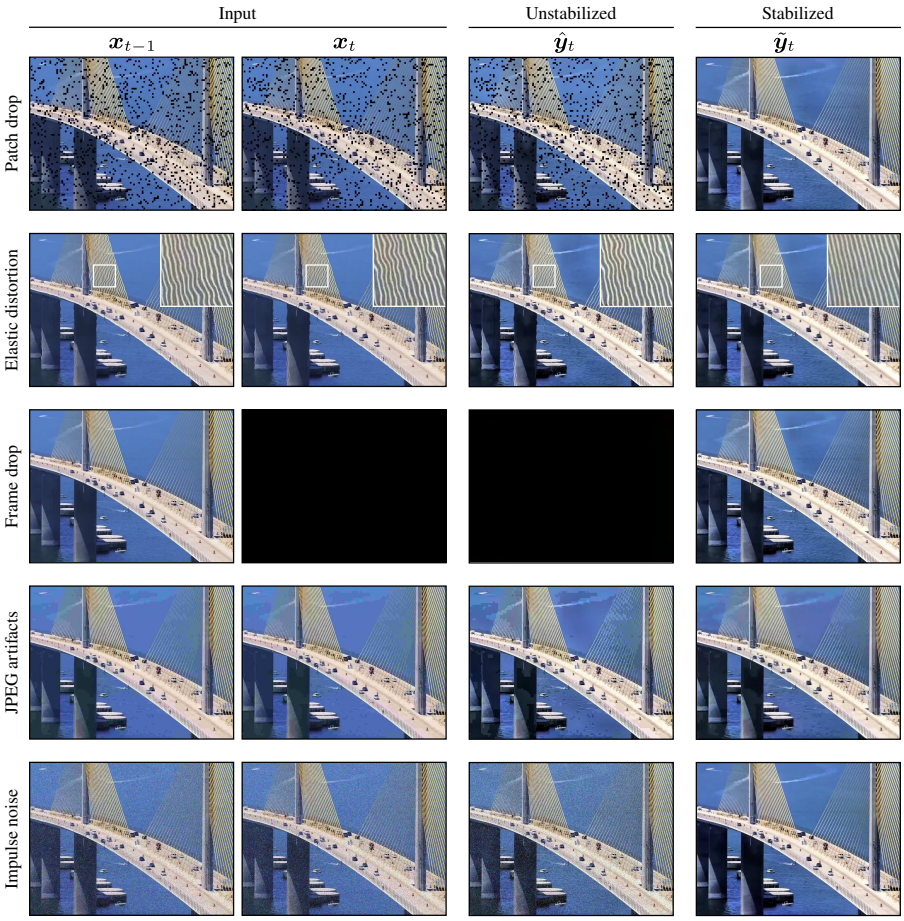


Figure D.5: **Image enhancement robustness.** The effect of stabilization for image enhancement (HDRNet) under various image corruptions. See Section 5.6.3.

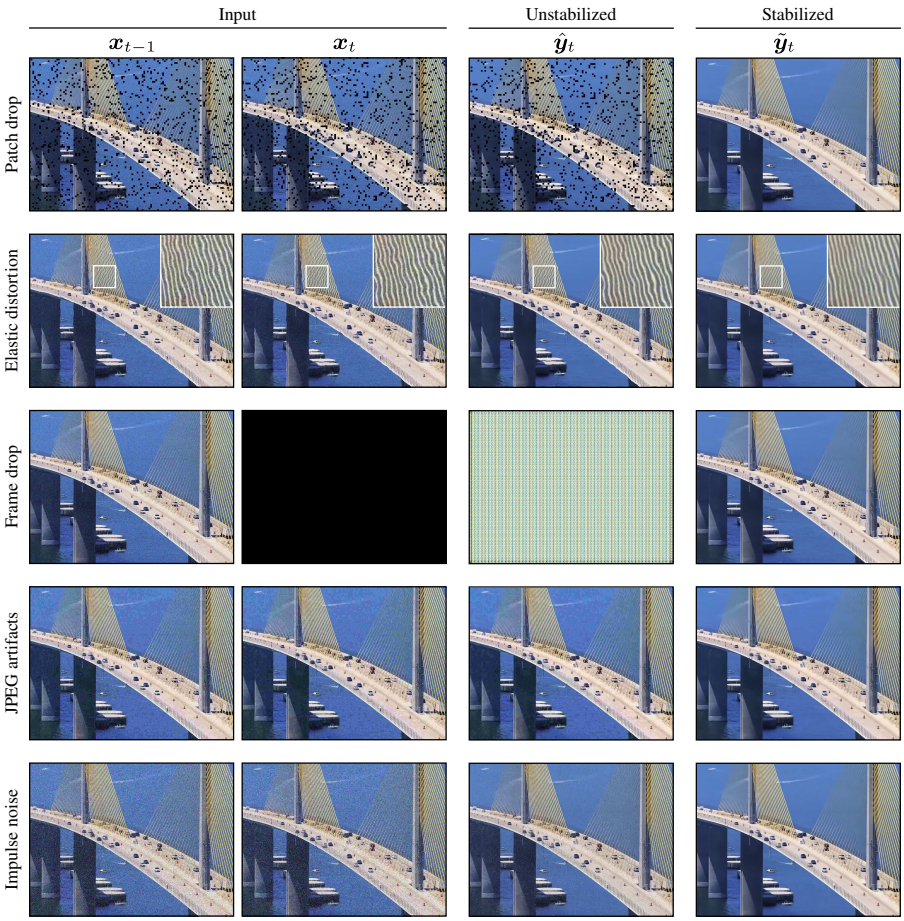


Figure D.6: **Denoising robustness.** The effect of stabilization for denoising (NAFNet) under various image corruptions. See Section 5.6.3.

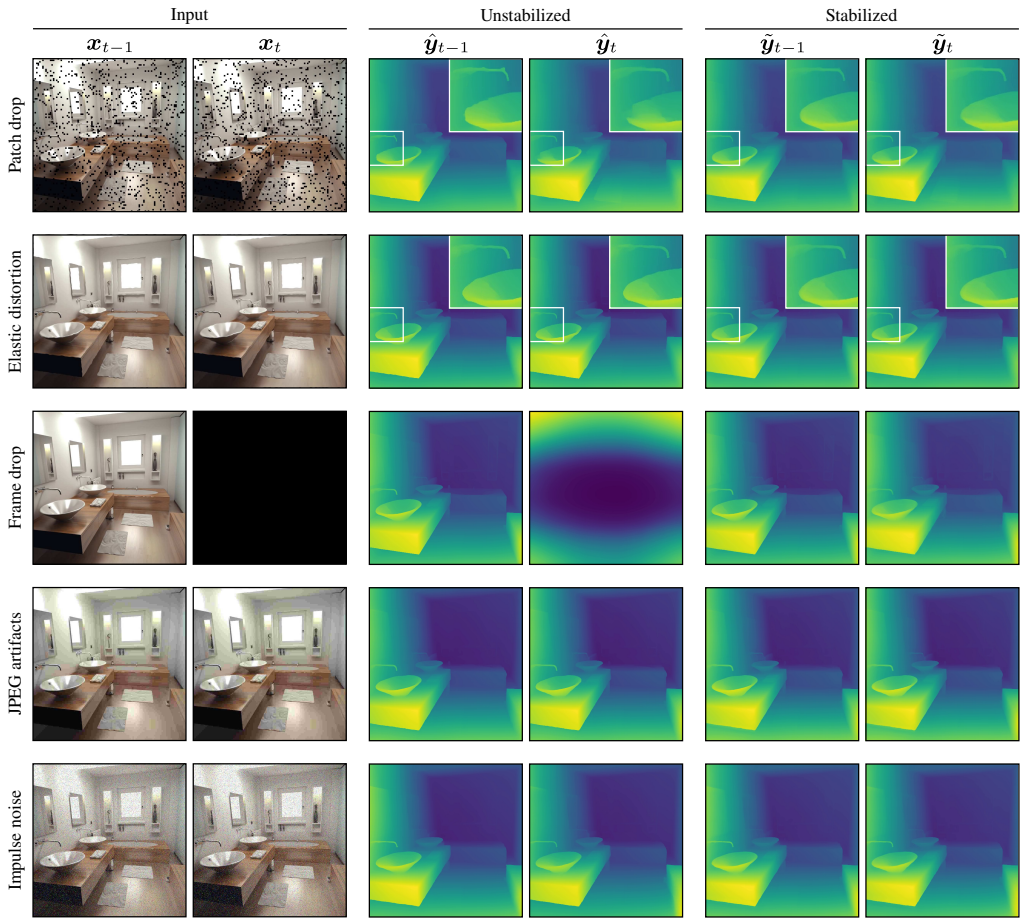


Figure D.7: **Depth estimation robustness.** The effect of stabilization for depth estimation (Depth Anything v2) under various image corruptions. Improvements are most prominent for the patch drop, elastic distortion, and frame drop corruptions. The base model already has reasonable robustness to JPEG artifacts and noise. See Section 5.6.3.

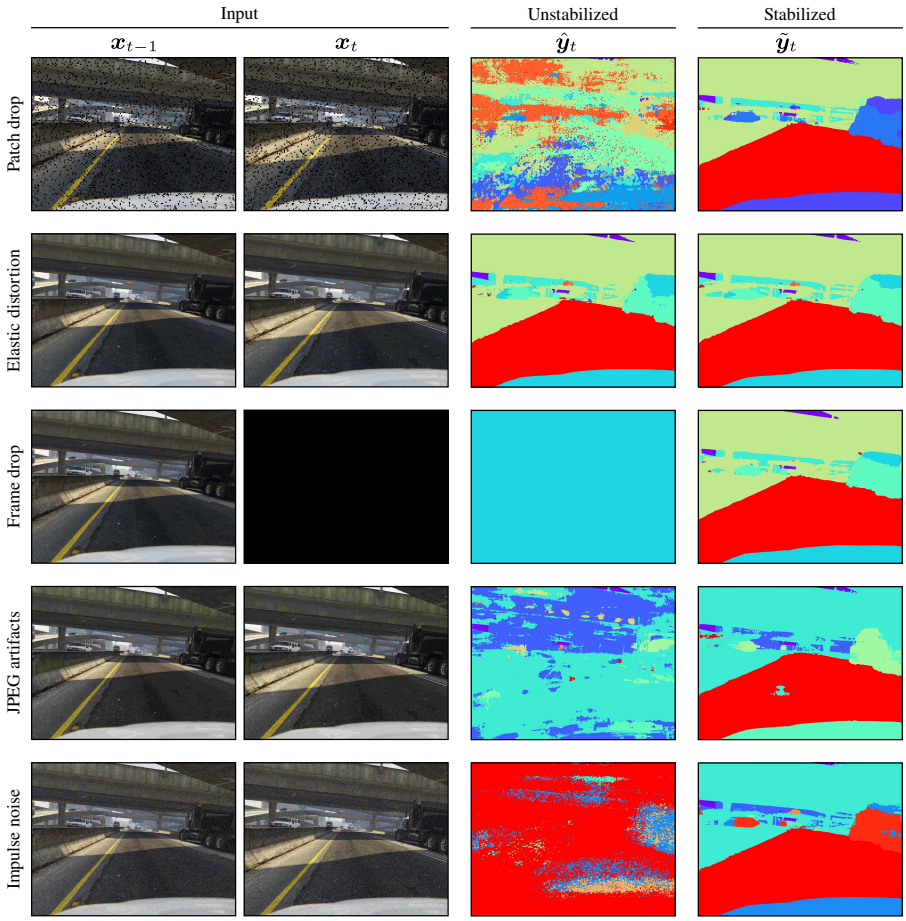


Figure D.8: **Segmentation robustness.** The effect of stabilization for segmentation (DeepLabv3+) under various image corruptions. See Section D.6.2. Note that the method we use to generate the color map may cause color-class correspondences to vary across rows.

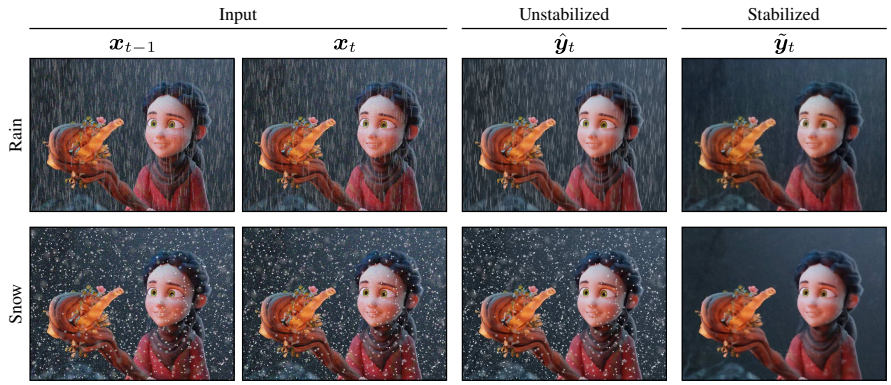


Figure D.9: **Adverse weather robustness.** The effect of stabilization for denoising (NAFNet) under weather corruptions on the RobustSpring dataset. See Section 5.6.4.

D.6.9 Tables

Table D.3 provides complete results for image enhancement (Figure 5.4). Tables D.4 and D.5 contain complete results for denoising (Figures 5.5 and D.3).

		Moderate intensity ($\alpha = 0.5$)			High intensity ($\alpha = 0.25$)		
Method	Strength	Instability	PSNR	SSIM	Instability	PSNR	SSIM
Gaussian	$\mu = 0.50$	29.46	30.70	0.917	36.26	25.73	0.852
Gaussian	$\mu = 1.00$	22.23	29.26	0.894	27.20	24.90	0.824
Gaussian	$\mu = 2.00$	16.49	27.38	0.848	19.99	23.65	0.773
Gaussian	$\mu = 3.00$	13.34	26.18	0.811	16.09	22.79	0.732
Gaussian	$\mu = 4.00$	11.29	25.33	0.782	13.57	22.15	0.700
Gaussian	$\mu = 6.00$	8.76	24.18	0.739	10.49	21.26	0.654
Output fixed	$\beta = 0.99$	32.74	30.92	0.919	40.38	25.84	0.856
Output fixed	$\beta = 0.98$	32.42	30.91	0.919	39.99	25.83	0.856
Output fixed	$\beta = 0.95$	31.50	30.87	0.919	38.82	25.81	0.855
Output fixed	$\beta = 0.90$	30.00	30.76	0.917	36.94	25.76	0.852
Output fixed	$\beta = 0.80$	27.16	30.40	0.912	33.37	25.57	0.846
Output fixed	$\beta = 0.60$	21.80	29.26	0.893	26.65	24.91	0.824
Simple fixed	$\beta = 0.99$	32.81	30.92	0.919	40.64	25.84	0.856
Simple fixed	$\beta = 0.98$	32.55	30.90	0.919	40.47	25.83	0.855
Simple fixed	$\beta = 0.95$	31.61	30.82	0.918	39.67	25.77	0.853
Simple fixed	$\beta = 0.90$	29.76	30.59	0.914	37.59	25.62	0.848
Simple fixed	$\beta = 0.80$	25.88	29.90	0.903	32.57	25.20	0.833
Simple fixed	$\beta = 0.60$	19.28	28.10	0.865	23.85	23.99	0.788
Simple learned	$\lambda = 0.1$	32.44	30.92	0.920	39.58	25.84	0.856
Simple learned	$\lambda = 0.2$	31.34	30.87	0.919	37.64	25.79	0.854
Simple learned	$\lambda = 0.4$	28.96	30.66	0.917	33.74	25.60	0.849
Simple learned	$\lambda = 0.8$	23.60	29.75	0.903	26.43	24.90	0.825
Controlled	$\lambda = 0.1$	31.78	31.25	0.922	38.31	26.26	0.865
Controlled	$\lambda = 0.2$	30.57	31.13	0.920	36.11	26.11	0.859
Controlled	$\lambda = 0.4$	28.00	30.86	0.917	31.98	25.92	0.855
Controlled	$\lambda = 0.8$	21.79	29.85	0.901	24.64	25.17	0.824
Spatial	$\lambda = 0.1$	32.38	32.81	0.934	40.32	27.97	0.891
Spatial	$\lambda = 0.2$	31.29	32.70	0.933	38.29	27.79	0.890
Spatial	$\lambda = 0.4$	28.66	32.30	0.928	34.22	27.45	0.881
Spatial	$\lambda = 0.8$	22.31	31.00	0.910	25.61	26.27	0.849

Table D.3: **Image enhancement results.** These results correspond to the experiments in Section 5.6.1 (Figure 5.4). We additionally include results for simple Gaussian smoothing of the output, where μ is the standard deviation of the smoothing kernel.

		NFS moderate ($\sigma = 0.1$)			NFS strong ($\sigma = 0.2$)		
Method	Strength	Instability	PSNR	SSIM	Instability	PSNR	SSIM
Gaussian	$\mu = 0.50$	22.87	36.77	0.945	24.50	33.52	0.903
Gaussian	$\mu = 1.00$	16.33	35.41	0.940	16.67	33.10	0.904
Gaussian	$\mu = 2.00$	11.90	32.96	0.920	11.77	31.64	0.889
Gaussian	$\mu = 3.00$	9.63	31.31	0.900	9.45	30.43	0.874
Gaussian	$\mu = 4.00$	8.18	30.15	0.883	8.01	29.50	0.860
Gaussian	$\mu = 6.00$	6.39	28.60	0.857	6.25	28.17	0.837
Output fixed	$\beta = 0.99$	25.93	36.67	0.943	28.18	33.30	0.900
Output fixed	$\beta = 0.98$	25.64	36.70	0.943	27.83	33.33	0.900
Output fixed	$\beta = 0.95$	24.79	36.76	0.944	26.81	33.40	0.901
Output fixed	$\beta = 0.90$	23.43	36.79	0.945	25.20	33.50	0.902
Output fixed	$\beta = 0.80$	20.91	36.58	0.945	22.22	33.54	0.904
Output fixed	$\beta = 0.60$	16.37	35.34	0.940	16.97	33.07	0.903
Simple fixed	$\beta = 0.99$	27.40	35.95	0.927	32.19	32.22	0.847
Simple fixed	$\beta = 0.98$	30.21	34.67	0.885	39.59	30.51	0.741
Simple fixed	$\beta = 0.95$	40.44	31.30	0.734	63.31	26.52	0.491
Simple fixed	$\beta = 0.90$	53.94	28.14	0.565	91.84	23.11	0.322
Simple fixed	$\beta = 0.80$	65.09	25.62	0.437	114.95	20.51	0.226
Simple fixed	$\beta = 0.60$	56.70	24.89	0.405	100.92	19.83	0.205
Simple learned	$\lambda = 0.1$	22.98	36.82	0.947	22.34	33.58	0.908
Simple learned	$\lambda = 0.2$	21.80	36.74	0.947	20.58	33.51	0.908
Simple learned	$\lambda = 0.4$	19.46	36.36	0.946	17.63	33.21	0.907
Simple learned	$\lambda = 0.8$	15.38	35.00	0.939	13.48	32.24	0.898
Controlled	$\lambda = 0.1$	22.21	37.51	0.952	21.41	34.36	0.916
Controlled	$\lambda = 0.2$	21.28	37.41	0.951	20.21	34.30	0.915
Controlled	$\lambda = 0.4$	19.01	37.00	0.949	17.24	33.93	0.912
Controlled	$\lambda = 0.8$	14.61	35.76	0.942	12.95	33.05	0.903
Spatial	$\lambda = 0.1$	22.13	37.65	0.953	21.08	34.52	0.917
Spatial	$\lambda = 0.2$	21.10	37.57	0.952	19.80	34.45	0.917
Spatial	$\lambda = 0.4$	18.94	37.15	0.950	16.99	34.06	0.913
Spatial	$\lambda = 0.8$	14.25	34.94	0.931	11.84	32.53	0.895

Table D.4: **Denoising results, part 1/2.** These results correspond to the experiments in Section 5.6.2 (Figure 5.5). We also include SSIM and results for simple Gaussian smoothing of the output, where μ is the standard deviation of the smoothing kernel. See Table D.5 for the second half of the data.

		NFS extreme ($\sigma = 0.6$)			DAVIS ($\sigma = 40/255$)		
Method	Strength	Instability	PSNR	SSIM	Instability	PSNR	SSIM
Gaussian	$\mu = 0.50$	30.23	27.98	0.794	105.62	30.67	0.864
Gaussian	$\mu = 1.00$	18.55	28.26	0.804	70.68	26.18	0.795
Gaussian	$\mu = 2.00$	11.82	28.02	0.803	45.38	23.22	0.717
Gaussian	$\mu = 3.00$	9.12	27.59	0.797	34.27	21.92	0.677
Gaussian	$\mu = 4.00$	7.59	27.17	0.790	27.95	21.15	0.653
Gaussian	$\mu = 6.00$	5.84	26.45	0.778	21.01	20.25	0.626
Output fixed	$\beta = 0.99$	35.64	27.72	0.787	121.46	31.65	0.873
Output fixed	$\beta = 0.98$	35.13	27.75	0.788	119.95	31.64	0.872
Output fixed	$\beta = 0.95$	33.65	27.83	0.790	115.50	31.48	0.871
Output fixed	$\beta = 0.90$	31.30	27.94	0.793	108.36	30.96	0.867
Output fixed	$\beta = 0.80$	26.98	28.12	0.798	94.92	29.54	0.853
Output fixed	$\beta = 0.60$	19.53	28.27	0.804	70.52	26.60	0.810
Simple fixed	$\beta = 0.99$	45.83	26.35	0.616	120.92	31.14	0.844
Simple fixed	$\beta = 0.98$	64.92	24.36	0.415	121.40	30.05	0.777
Simple fixed	$\beta = 0.95$	119.74	20.09	0.191	130.25	26.85	0.581
Simple fixed	$\beta = 0.90$	180.81	16.56	0.105	148.51	23.76	0.415
Simple fixed	$\beta = 0.80$	230.73	13.86	0.067	164.16	21.26	0.309
Simple fixed	$\beta = 0.60$	205.99	12.97	0.059	137.79	20.09	0.270
Simple learned	$\lambda = 0.1$	20.24	28.29	0.812	118.48	31.63	0.873
Simple learned	$\lambda = 0.2$	17.73	28.27	0.813	114.84	31.49	0.871
Simple learned	$\lambda = 0.4$	14.25	28.13	0.812	107.17	30.94	0.865
Simple learned	$\lambda = 0.8$	10.21	27.69	0.806	80.09	27.89	0.821
Controlled	$\lambda = 0.1$	19.47	29.22	0.824	117.58	31.85	0.876
Controlled	$\lambda = 0.2$	17.26	29.15	0.824	113.44	31.69	0.873
Controlled	$\lambda = 0.4$	13.67	28.92	0.822	104.21	31.02	0.864
Controlled	$\lambda = 0.8$	10.12	28.45	0.813	77.15	27.86	0.815
Spatial	$\lambda = 0.1$	15.20	22.00	0.665	117.57	31.86	0.876
Spatial	$\lambda = 0.2$	16.55	21.73	0.637	113.49	31.70	0.874
Spatial	$\lambda = 0.4$	9.09	22.28	0.692	104.26	31.05	0.865
Spatial	$\lambda = 0.8$	11.66	22.45	0.669	81.86	28.15	0.824

Table D.5: **Denoising results, part 2/2.** These results correspond to the experiments in Section 5.6.2 (Figure 5.5). We additionally include SSIM and results for simple Gaussian smoothing of the output, where μ is the standard deviation of the smoothing kernel. See Table D.4 for the first half of the data.

D.7 Licenses and Copyright

Code. We use our own implementation of HDRNet. For NAFNet, we use the authors' code (MIT license). For Depth Anything, we use the depth-anything/Depth-Anything-V2-Small-hf HuggingFace module (available under an Apache 2.0 license). VisionSim is released under the MIT license. All scenes are licensed under a Creative Commons variant; see this Google Drive folder for attributions and further license details.

Datasets and assets. We were unable to find license information for the Need for Speed dataset. DAVIS uses the BSD license. We use frames from the following YouTube videos in our figures:

- https://www.youtube.com/watch?v=ANeMCOpx_84
- https://www.youtube.com/watch?v=HZ8VF0EdITk
- https://www.youtube.com/watch?v=MPZb9EQ3Wjs
- https://www.youtube.com/watch?v=obSH5F2DYvk

Bibliography

[1] Ryan Prescott Adams and David J. C. MacKay. Bayesian online changepoint detection. arXiv, 2007.

[2] Naveed Akhtar, Ajmal Mian, Navid Kardan, and Mubarak Shah. Advances in adversarial attacks and defenses in computer vision: A survey. *IEEE Access*, 9:155161–155196, 2021.

[3] Filipp Akopyan, Jun Sawada, Andrew Cassidy, Rodrigo Alvarez-Icaza, John Arthur, Paul Merolla, Nabil Imam, Yutaka Nakamura, Pallab Datta, Gi-Joon Nam, Brian Taba, Michael Beakes, Bernard Brezzo, Jente B. Kuang, Rajit Manohar, William P. Risk, Bryan Jackson, and Dharmendra S. Modha. TrueNorth: Design and tool flow of a 65 mW 1 million neuron programmable neurosynaptic chip. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(10):1537–1557, October 2015.

[4] Réda Alami, Odalric Maillard, and Raphael Féraud. Restarted bayesian online change-point detector achieves optimal detection delay. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 211–221, 2020.

[5] Ignacio Alzugaray and Margarita Chli. Asynchronous corner detection and tracking for event cameras in real time. *IEEE Robotics and Automation Letters*, 3(4):3177–3184, 2018.

[6] Mykhaylo Andriluka, Leonid Pishchulin, Peter Gehler, and Bernt Schiele. 2D human pose estimation: New benchmark and state of the art analysis. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3686–3693, 2014.

[7] Andrei Ardelean. *Computational Imaging SPAD Cameras*. PhD thesis, École polytechnique fédérale de Lausanne, 2023.

[8] Anurag Arnab, Mostafa Dehghani, Georg Heigold, Chen Sun, Mario Lučić, and Cordelia Schmid. ViViT: A video vision transformer. In *Proceedings of the International Conference on Computer Vision (ICCV)*, pages 6836–6846, October 2021.

[9] Aharon Azulay, Tavi Halperin, Orestis Vantzos, Nadav Bornstein, and Ofir Bibi. Temporally stable video segmentation without video annotations. In *Proceedings of the Winter Conference on Applications of Computer Vision (WACV)*, pages 3449–3458, January 2022.

[10] Simon Baker, Daniel Scharstein, J. P. Lewis, Stefan Roth, Michael J. Black, and Richard Szeliski. A database and evaluation methodology for optical flow. *International Journal of Computer Vision (IJCV)*, 92(1):1–31, March 2011.

[11] Souptik Barua, Yoshitaka Miyatani, and Ashok Veeraraghavan. Direct face detection and video reconstruction from event cameras. In *Proceedings of the Winter Conference on Applications of Computer Vision (WACV)*, pages 1–9, 2016.

[12] Philipp Benz, Chaoning Zhang, Adil Karjauv, and In So Kweon. Revisiting batch normalization for improving corruption robustness. In *Proceedings of the Winter Conference on Applications of Computer Vision (WACV)*, pages 494–503, January 2021.

[13] Raphael Berner, Christian Brandli, Minhao Yang, S-C Liu, and Tobi Delbruck. A 240x180 120 dB 10 mW 12 μs-latency sparse output vision sensor for mobile applications. In *Proceedings of the International Image Sensors Workshop*, pages 41–44, 2013.

[14] Gedas Bertasius, Lorenzo Torresani, and Jianbo Shi. Object detection in video with spatiotemporal sampling networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

[15] Gedas Bertasius, Heng Wang, and Lorenzo Torresani. Is space-time attention all you need for video understanding? In *Proceedings of the International Conference on Machine Learning (ICML)*, Proceedings of Machine Learning Research, 2021.

[16] Anthony Bisulco, Fernando Cladera Ojeda, Volkan Isler, and Daniel D. Lee. Fast motion understanding with spatiotemporal neural networks and dynamic vision sensors. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, pages 14098–14104, 2021.

[17] Daniel Bolya, Cheng-Yang Fu, Xiaoliang Dai, Peizhao Zhang, Christoph Feichtenhofer, and Judy Hoffman. Token merging: Your ViT but faster. arXiv, 2022.

[18] Nicolas Bonneel, James Tompkin, Kalyan Sunkavalli, Deqing Sun, Sylvain Paris, and Hanspeter Pfister. Blind video temporal consistency. *Transactions on Graphics (TOG)*, 34(6), 2015.

[19] Assim Boukhayma, Arnaud Peizerat, and Christian Enz. A sub-0.5 electron read noise VGA image sensor in a standard CMOS process. *IEEE Journal of Solid-State Circuits*, 2016.

[20] Christian Brandli, Raphael Berner, Minhao Yang, Shih-Chii Liu, and Tobi Delbruck. A 240x180 130 dB 3 μs latency global shutter spatiotem-

poral vision sensor. *IEEE Journal of Solid-State Circuits*, 49(10):2333–2341, 2014.

[21] Christian Brandli, Lorenz Muller, and Tobi Delbruck. Real-time, high-speed video decompression using a frame-and event-based DAVIS sensor. In *2014 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 686–689, 2014.

[22] Daniel J. Butler, Jonas Wulff, Garrett B. Stanley, and Michael J. Black. A naturalistic open source movie for optical flow evaluation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 611–625, 2012.

[23] BuzzFarmers. Dog is running, 2011. Accessed March 2023, CC BY 2.0 license.

[24] Zhaowei Cai and Nuno Vasconcelos. Cascade R-CNN: High quality object detection and instance segmentation. *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 43(5):1483–1498, 2021.

[25] Víctor Campos, Brendan Jou, Xavier Giró-i Nieto, Jordi Torres, and Shih-Fu Chang. Skip RNN: Learning to skip state updates in recurrent neural networks. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018.

[26] Marco Cannici, Marco Ciccone, Andrea Romanoni, and Matteo Matteucci. Asynchronous convolutional networks for object detection in neuromorphic cameras. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pages 1656–1665, June 2019.

[27] Alfredo Canziani, Adam Paszke, and Eugenio Culurciello. An analysis of deep neural network models for practical applications. arXiv, 2016.

[28] Zhe Cao, Tomas Simon, Shih-En Wei, and Yaser Sheikh. Realtime multi-person 2D pose estimation using part affinity fields. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7291–7299, 2017.

[29] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with Transformers. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 213–229, 2020.

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

[31] Andrea Censi and Davide Scaramuzza. Low-latency event-based visual odometry. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, pages 703–710, 2014.

[32] Ya-Liang Chang, Zhe Yu Liu, Kuan-Ying Lee, and Winston Hsu. Learnable gated temporal shift module for deep video inpainting. arXiv, 2019.

[33] Dongdong Chen, Jing Liao, Lu Yuan, Nenghai Yu, and Gang Hua. Coherent online video style transfer. In *Proceedings of the International Conference on Computer Vision (ICCV)*, October 2017.

[34] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation. arXiv, 2017.

[35] Liangyu Chen, Xiaojie Chu, Xiangyu Zhang, and Jian Sun. Simple baselines for image restoration. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 17–33, 2022.

[36] Tian Qi Chen and Mark Schmidt. Fast patch-based style transfer of arbitrary style. arXiv, 2016.

[37] Xinghao Chen, Yiman Zhang, Yunhe Wang, Han Shu, Chunjing Xu, and Chang Xu. Optical flow distillation: Towards efficient and stable video style transfer. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 614–630, 2020.

[38] Yihong Chen, Yue Cao, Han Hu, and Liwei Wang. Memory enhanced global-local aggregation for video object detection. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[39] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. arXiv, 2019.

[40] Ting-Wu Chin, Ruizhou Ding, and Diana Marculescu. AdaScale: Towards real-time video object detection using adaptive scaling. In *Proceedings of Machine Learning and Systems*, volume 1, pages 431–441, 2019.

[41] Krzysztof Marcin Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Quincy Davis, Afroz Mohiuddin, Lukasz Kaiser, David Benjamin Belanger, Lucy J. Colwell, and Adrian Weller. Rethinking attention with Performers. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

[42] Matthew Cook, Luca Gugelmann, Florian Jug, Christoph Krautz, and Angelika Steger. Interacting maps for fast visual interpretation. In *The International Joint Conference on Neural Networks*, pages 770–776, 2011.

[43] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The Cityscapes dataset for semantic urban scene understanding. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[44] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. BinaryConnect: Training deep neural networks with binary weights during propagations. *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 28:3123–3131, 2015.

[45] Dima Damen, Hazel Doughty, Giovanni Maria Farinella, Sanja Fidler, Antonino Furnari, Evangelos Kazakos, Davide Moltisanti, Jonathan Munro, Toby Perrett, Will Price, and Michael Wray. Scaling egocentric vision: The EPIC-KITCHENS dataset. In *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

[46] Mike Davies, Narayan Srinivasa, Tsung-Han Lin, Gautham Chinya, Yongqiang Cao, Sir Harsha Choday, Georgios Dimou, Prasad Joshi, Nabil Imam, Shweta Jain, Yunyun Liao, Chit-Kwan Lin, Andrew Lines, Ruokun Liu, Deepak Mathaikutty, Steven McCoy, Arnab Paul, Jonathan Tse, Gurugahanathan Venkataramanan, Yi-Hsin Weng, Andreas Wild, Yoonseok Yang, and Hong Wang. Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro*, 38(1):82–99, January 2018.

[47] Yingying Deng, Fan Tang, Weiming Dong, Haibin Huang, Chongyang Ma, and Changsheng Xu. Arbitrary video style transfer via multi-channel correlation. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pages 1210–1217, May 2021.

[48] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani,

Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

[49] Nathan Drenkow, Numair Sani, Ilya Shpitser, and Mathias Unberath. A systematic review of robustness in deep learning for computer vision: Mind the gap? arXiv, 2022.

[50] Matthew Dutson, Yin Li, and Mohit Gupta. Event neural networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 276–293, 2022.

[51] Matthew Dutson, Yin Li, and Mohit Gupta. Eventful Transformers: Leveraging temporal redundancy in vision transformers. In *Proceedings of the International Conference on Computer Vision (ICCV)*, pages 16911–16923, October 2023.

[52] Matthew Dutson, Yin Li, and Mohit Gupta. Spike-based anytime perception. In *Proceedings of the Winter Conference on Applications of Computer Vision (WACV)*, pages 5294–5304, January 2023.

[53] Neale A. W. Dutton, Istvan Gyongy, Luca Parmesan, Salvatore Gnecchi, Neil Calder, Bruce R. Rae, Sara Pellegrini, Linsay A. Grant, and Robert K. Henderson. A SPAD-based QVGA image sensor for single-photon counting and quanta imaging. *IEEE Transactions on Electron Devices*, 63(1):189–196, January 2016.

[54] Haoqi Fan, Bo Xiong, Karttikeya Mangalam, Yanghao Li, Zhicheng Yan, Jitendra Malik, and Christoph Feichtenhofer. Multiscale vision transformers. In *Proceedings of the International Conference on Computer Vision (ICCV)*, pages 6824–6835, October 2021.

[55] Gongfan Fang. Pretrained DeepLabv3 and DeepLabv3+ for Pascal VOC and Cityscapes. GitHub, 2022.

[56] Mohsen Fayyaz, Soroush Abbasi Koohpayegani, Farnoush Rezaei Jafari, Sunando Sengupta, Hamid Reza Vaezi Joze, Eric Sommerlade, Hamed Pirsiavash, and Juergen Gall. Adaptive token sampling for efficient vision transformers. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 396–414, 2022.

[57] Christoph Feichtenhofer, Haoqi Fan, Jitendra Malik, and Kaiming He. SlowFast networks for video recognition. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6202–6211, 2019.

[58] Michael Figurnov, Maxwell D. Collins, Yukun Zhu, Li Zhang, Jonathan Huang, Dmitry Vetrov, and Ruslan Salakhutdinov. Spatially adaptive computation time for residual networks. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[59] Chang Gao, Derun Gu, Fangjun Zhang, and Yizhou Yu. ReCoNet: Real-time coherent video style transfer network. In *Proceedings of the Asian Conference on Computer Vision (ACCV)*, pages 637–653, 2019.

[60] Wei Gao, Yijun Li, Yihang Yin, and Ming-Hsuan Yang. Fast video multi-style transfer. In *Proceedings of the Winter Conference on Applications of Computer Vision (WACV)*, March 2020.

[61] Yue Gao, Siqi Li, Yipeng Li, Yandong Guo, and Qionghai Dai. SuperFast: 200x video frame interpolation via event camera. *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 45(6):7764–7780, 2023.

[62] Daniel Gehrig, Henri Rebecq, Guillermo Gallego, and Davide Scaramuzza. EKLT: Asynchronous photometric feature tracking using events

and frames. *International Journal of Computer Vision (IJCV)*, 128(3):601–618, 2020.

[63] Mathias Gehrig, Mario Millhäusler, Daniel Gehrig, and Davide Scaramuzza. E-RAFT: Dense optical flow from event cameras. In *2021 International Conference on 3D Vision (3DV)*, pages 197–206, 2021.

[64] Michaël Gharbi, Jiawen Chen, Jonathan T. Barron, Samuel W. Hasinoff, and Frédo Durand. Deep bilateral learning for real-time image enhancement. *Transactions on Graphics (TOG)*, 36(4), July 2017.

[65] Amir Ghodrati, Babak Ehteshami Bejnordi, and Amirhossein Habibian. Frameexit: Conditional early exiting for efficient video recognition. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 15608–15618, 2021.

[66] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. arXiv, 2015.

[67] Saurabh Goyal, Anamitra Roy Choudhury, Saurabh Raje, Venkatesan Chakaravarthy, Yogish Sabharwal, and Ashish Verma. PoWER-BERT: Accelerating BERT inference via progressive word-vector elimination. In *Proceedings of the International Conference on Machine Learning (ICML)*, volume 119 of *Proceedings of Machine Learning Research*, pages 3690–3699, July 2020.

[68] Rui Graça, Brian McReynolds, and Tobi Delbruck. Shining light on the DVS pixel: A tutorial and discussion about biasing and optimization. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pages 4045–4053, June 2023.

[69] Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. In *First Conference on Language Modeling*, 2024.

[70] Qipeng Guo, Xipeng Qiu, Pengfei Liu, Yunfan Shao, Xiangyang Xue, and Zheng Zhang. Star-transformer. arXiv, 2019.

[71] Agrim Gupta, Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Characterizing and improving stability in neural style transfer. In *Proceedings of the International Conference on Computer Vision (ICCV)*, October 2017.

[72] Istvan Gyongy, Neale A. W. Dutton, and Robert K. Henderson. Single-photon tracking for high-speed vision. *Sensors*, 18(2):323, 2018.

[73] Amirhossein Habibian, Davide Abati, Taco S. Cohen, and Babak Ehteshami Bejnordi. Skip-convolutions for efficient video processing. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2695–2704, June 2021.

[74] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 28:1135–1143, 2015.

[75] Chris Harris, Mike Stephens, et al. A combined corner and edge detector. In *Alvey vision conference*, volume 15, pages 10–5244, 1988.

[76] Samuel W. Hasinoff, Dillon Sharlet, Ryan Geiss, Andrew Adams, Jonathan T. Barron, Florian Kainz, Jiawen Chen, and Marc Levoy. Burst photography for high dynamic range and low-light imaging on mobile cameras. *Transactions on Graphics (TOG)*, 35(6):1–12, November 2016.

[77] Babak Hassibi and David Stork. Second order derivatives for network pruning: Optimal brain surgeon. *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 5:164–171, 1992.

[78] Botao He, Haojia Li, Siyuan Wu, Dong Wang, Zhiwei Zhang, Qianli Dong, Chao Xu, and Fei Gao. Fast-dynamic-vision: Detection and tracking dynamic objects with event and depth sensing. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3071–3078, 2021.

[79] Kaiming He, Georgia Gkioxari, Piotr Dollar, and Ross Girshick. Mask R-CNN. In *Proceedings of the International Conference on Computer Vision (ICCV)*, October 2017.

[80] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[81] Dan Hendrycks and Thomas Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019.

[82] Javier Hidalgo-Carrió, Guillermo Gallego, and Davide Scaramuzza. Event-aided direct sparse odometry. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5781–5790, June 2022.

[83] Yasunobu Hitomi, Jinwei Gu, Mohit Gupta, Tomoo Mitsunaga, and Shree K. Nayar. Video from a single coded exposure photograph using a learned over-complete dictionary. In *Proceedings of the International Conference on Computer Vision (ICCV)*, pages 287–294, 2011.

[84] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[85] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam.

MobileNets: Efficient convolutional neural networks for mobile vision applications. arXiv, 2017.

[86] Yuhuang Hu, Shih-Chii Liu, and Tobi Delbruck. v2e: From video frames to realistic DVS events. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pages 1312–1321, June 2021.

[87] Gao Huang, Danlu Chen, Tianhong Li, Felix Wu, Laurens Van Der Maaten, and Kilian Q Weinberger. Multi-scale dense networks for resource efficient image classification. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018.

[88] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q Weinberger. Deep networks with stochastic depth. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 646–661, 2016.

[89] Haozhi Huang, Hao Wang, Wenhan Luo, Lin Ma, Wenhao Jiang, Xiaolong Zhu, Zhifeng Li, and Wei Liu. Real-time neural style transfer for videos. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[90] Jing Huang, Menghan Guo, and Shoushun Chen. A dynamic vision sensor with direct logarithmic output and full-frame picture-on-demand. In *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–4, 2017.

[91] Zhewei Huang, Tianyuan Zhang, Wen Heng, Boxin Shi, and Shuchang Zhou. Real-time intermediate flow estimation for video frame interpolation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2022.

[92] Kyuyeon Hwang and Wonyong Sung. Fixed-point feedforward deep neural network design using weights +1, 0, and -1. In *2014 IEEE Workshop on Signal Processing Systems (SiPS)*, pages 1–6, 2014.

[93] Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and 0.5MB model size. arXiv, 2016.

[94] Jorge Igual. Photographic noise performance measures based on raw files analysis of consumer cameras. *Electronics*, 8(11):1284, 2019.

[95] Atul Ingle, Trevor Seets, Mauro Buttafava, Shantanu Gupta, Alberto Tosi, Mohit Gupta, and Andreas Velten. Passive inter-photon imaging. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2021.

[96] Atul Ingle, Andreas Velten, and Mohit Gupta. High Flux Passive Imaging With Single-Photon Sensors. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[97] Kiyotaka Iwabuchi, Yusuke Kameda, and Takayuki Hamamoto. Image quality improvements based on motion-based deblurring for single-photon imaging. *IEEE Access*, 9:30080–30094, 2021.

[98] Samvit Jain, Xin Wang, and Joseph E. Gonzalez. Accel: A corrective fusion network for efficient semantic segmentation on video. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8866–8875, 2019.

[99] Sebastian Jaszczur, Aakanksha Chowdhery, Afroz Mohiuddin, LUKASZ KAISER, Wojciech Gajewski, Henryk Michalewski, and Jonni Kanerva. Sparse is enough in scaling transformers. In *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, volume 34, pages 9895–9907, 2021.

[100] H. Jhuang, J. Gall, S. Zuffi, C. Schmid, and M. J. Black. Towards understanding action recognition. In *Proceedings of the International Conference on Computer Vision (ICCV)*, pages 3192–3199, December 2013.

[101] Sacha Jungerman. VisionSim: A modular and extensible framework for distributed simulations with rich pixel-perfect ground truth annotations and realistic sensor emulation. https://visionsim.readthedocs.io, 2025. Accessed 2025-05-15.

[102] Kai Kang, Hongsheng Li, Tong Xiao, Wanli Ouyang, Junjie Yan, Xihui Liu, and Xiaogang Wang. Object detection in videos with tubelet proposal networks. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[103] Nikita Karaev, Ignacio Rocco, Benjamin Graham, Natalia Neverova, Andrea Vedaldi, and Christian Rupprecht. CoTracker: It is better to track together. arxiv, 2023.

[104] Kevin Karsch, Ce Liu, and Sing Bing Kang. Depth transfer: Depth extraction from video using non-parametric sampling. *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 36(11):2144–2158, 2014.

[105] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are RNNs: Fast autoregressive transformers with linear attention. In *Proceedings of the International Conference on Machine Learning (ICML)*, volume 119 of *Proceedings of Machine Learning Research*, pages 5156–5165, July 2020.

[106] Anthony Kay. Tesseract: an open-source optical character recognition engine. *Linux Journal*, 2007(159):2, 2007.

[107] Will Kay, Joao Carreira, Karen Simonyan, Brian Zhang, Chloe Hillier, Sudheendra Vijayanarasimhan, Fabio Viola, Tim Green, Trevor Back, Paul Natsev, Mustafa Suleyman, and Andrew Zisserman. The Kinetics human action video dataset. arXiv, 2017.

[108] Bingxin Ke, Dominik Narnhofer, Shengyu Huang, Lei Ke, Torben Peters, Katerina Fragkiadaki, Anton Obukhov, and Konrad Schindler. Video depth without video models. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7233–7243, June 2025.

[109] Onur Keleş, M. Akın Yılmaz, A. Murat Tekalp, Cansu Korkmaz, and Zafer Doğan. On the computation of psnr for a set of images or video. In *2021 Picture Coding Symposium (PCS)*, pages 1–5, 2021.

[110] Hamed Kiani Galoogahi, Ashton Fagg, Chen Huang, Deva Ramanan, and Simon Lucey. Need for Speed: A benchmark for higher frame rate object tracking. In *Proceedings of the International Conference on Computer Vision (ICCV)*, October 2017.

[111] Dahun Kim, Sanghyun Woo, Joon-Young Lee, and In So Kweon. Recurrent temporal aggregation framework for deep video inpainting. *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 42(5):1038–1052, 2020.

[112] Hanme Kim, Ankur Handa, Ryad Benosman, Sio-Hoi Ieng, and Andrew J. Davison. Simultaneous mosaicing and tracking with an event camera. *Proceedings of the British Machine Vision Conference (BMVC)*, 43:566–576, 2014.

[113] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. arXiv, 2017.

[114] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. arXiv, 2014.

[115] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C. Berg, Wan-Yen Lo, Piotr Dollar, and Ross Girshick. Segment anything. In *Proceedings of the International Conference on Computer Vision (ICCV)*, pages 4015–4026, October 2023.

[116] Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2020.

[117] Johannes Kopf, Xuejian Rong, and Jia-Bin Huang. Robust consistent video depth estimation. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1611–1621, June 2021.

[118] Wei-Sheng Lai, Jia-Bin Huang, Oliver Wang, Eli Shechtman, Ersin Yumer, and Ming-Hsuan Yang. Learning blind video temporal consistency. In *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

[119] Alfred Laugros, Alice Caplier, and Matthieu Ospici. Are adversarial robustness and common perturbation robustness independant attributes? In *Proceedings of the International Conference on Computer Vision (ICCV) Workshops*, October 2019.

[120] Martin Laurenzis, Emmanuel Bacher, Trevor Seets, Atul Ingle, Andreas Velten, and Frank Christnacher. Single photon flux imaging with subpixel resolution by motion compensation. In *Advanced Photon Counting Techniques*, volume 12512, pages 77–85, 2023.

[121] Martin Laurenzis, Trevor Seets, Emmanuel Bacher, Atul Ingle, and Andreas Velten. Comparison of super-resolution and noise reduc-

tion for passive single-photon imaging. *Journal of Electronic Imaging*, 31(3):033042–033042, 2022.

[122] Yann LeCun, John Denker, and Sara Solla. Optimal brain damage. *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2:598–605, 1989.

[123] Sungho Lee, Seoung Wug Oh, DaeYeun Won, and Seon Joo Kim. Copy-and-paste networks for deep video inpainting. In *Proceedings of the International Conference on Computer Vision (ICCV)*, October 2019.

[124] Chenyang Lei and Qifeng Chen. Fully automatic video colorization with self-regularization and diversity. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[125] Chenyang Lei, Yazhou Xing, and Qifeng Chen. Blind video temporal consistency via deep video prior. In *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, volume 33, pages 1083–1093, 2020.

[126] Dasong Li, Xiaoyu Shi, Yi Zhang, Ka Chun Cheung, Simon See, Xiaogang Wang, Hongwei Qin, and Hongsheng Li. A simple baseline for video restoration with grouped spatial-temporal shift. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9822–9832, June 2023.

[127] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient ConvNets. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.

[128] Haolong Li and Joerg Stueckler. Tracking 6-DoF object motion from events and frames. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, pages 14171–14177, 2021.

[129] Siyuan Li, Yue Luo, Ye Zhu, Xun Zhao, Yu Li, and Ying Shan. Enforcing temporal consistency in video depth estimation. In *Proceedings of the International Conference on Computer Vision (ICCV) Workshops*, pages 1145–1154, October 2021.

[130] Xueting Li, Sifei Liu, Jan Kautz, and Ming-Hsuan Yang. Learning linear transformations for fast image and video style transfer. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[131] Yanghao Li, Hanzi Mao, Ross Girshick, and Kaiming He. Exploring plain vision transformer backbones for object detection. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 280–296, 2022.

[132] Yawei Li, Babak Ehteshami Bejnordi, Bert Moons, Tijmen Blankevoort, Amirhossein Habibian, Radu Timofte, and Luc Van Gool. Spatio-temporal gated transformers for efficient video processing. In *Conference on Neural Information Processing Systems (NeurIPS) Workshops*, 2021.

[133] Yule Li, Jianping Shi, and Dahua Lin. Low-latency video semantic segmentation. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5997–6005, 2018.

[134] Jingyun Liang, Jiezhang Cao, Yuchen Fan, Kai Zhang, Rakesh Ranjan, Yawei Li, Radu Timofte, and Luc Van Gool. VRT: A video restoration transformer. *Transactions on Image Processing (TIP)*, 33:2171–2182, 2024.

[135] Youwei Liang, Chongjian G. E., Zhan Tong, Yibing Song, Jue Wang, and Pengtao Xie. EViT: Expediting vision transformers via token reorganizations. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2022.

[136] Patrick Lichtsteiner. 64x64 event-driven logarithmic temporal derivative silicon retina. In *Program 2003 IEEE Workshop on CCD and AIS*, 2003.

[137] Patrick Lichtsteiner, Christoph Posch, and Tobi Delbruck. A 128x128 120 dB 15 μs latency asynchronous temporal contrast vision sensor. *IEEE Journal of Solid-State Circuits*, 43(2):566–576, February 2008.

[138] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollar. Focal loss for dense object detection. In *Proceedings of the International Conference on Computer Vision (ICCV)*, pages 2980–2988, 2017.

[139] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: Common objects in context. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 740–755, 2014.

[140] Chang Liu, Yinpeng Dong, Wenzhao Xiang, Xiao Yang, Hang Su, Jun Zhu, Yuefeng Chen, Yuan He, Hui Xue, and Shibao Zheng. A comprehensive study on robustness of image classification models: Benchmarking and rethinking. *International Journal of Computer Vision (IJCV)*, 133:567–589, 2025.

[141] Mason Liu and Menglong Zhu. Mobile video object detection with temporally-aware feature maps. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[142] Songhua Liu, Tianwei Lin, Dongliang He, Fu Li, Meiling Wang, Xin Li, Zhengxing Sun, Qian Li, and Errui Ding. AdaAttN: Revisit attention mechanism in arbitrary neural style transfer. In *Proceedings of the International Conference on Computer Vision (ICCV)*, pages 6649–6658, October 2021.

[143] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: Single shot MultiBox detector. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 21–37, 2016.

[144] Yihao Liu, Hengyuan Zhao, Kelvin C. K. Chan, Xintao Wang, Chen Change Loy, Yu Qiao, and Chao Dong. Temporally consistent video colorization with deep feature propagation and self-regularization learning. *Computational Visual Media*, 10:375–395, 2024.

[145] Yuhao Liu, Felipe Gutierrez-Barragan, Atul Ingle, Mohit Gupta, and Andreas Velten. Single-photon camera guided extreme dynamic range imaging. In *Proceedings of the Winter Conference on Applications of Computer Vision (WACV)*, pages 1575–1585, January 2022.

[146] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the International Conference on Computer Vision (ICCV)*, pages 10012–10022, October 2021.

[147] Ilya Loshchilov and Frank Hutter. SGDR: Stochastic gradient descent with warm restarts. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.

[148] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019.

[149] Jiachen Lu, Jinghan Yao, Junge Zhang, Xiatian Zhu, Hang Xu, Weiguo Gao, Chunjing XU, Tao Xiang, and Li Zhang. SOFT: Softmax-free transformer with linear complexity. In *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, volume 34, pages 21297–21309, 2021.

[150] Xuan Luo, Jia-Bin Huang, Richard Szeliski, Kevin Matzen, and Johannes Kopf. Consistent video depth estimation. *Transactions on Graphics (TOG)*, 39(4), August 2020.

[151] Sizhuo Ma, Shantanu Gupta, Arin C. Ulku, Claudio Bruschini, Edoardo Charbon, and Mohit Gupta. Quanta burst photography. *Transactions on Graphics (TOG)*, 39(4):1–16, July 2020.

[152] Sizhuo Ma, Paul Mos, Edoardo Charbon, and Mohit Gupta. Burst vision using single-photon cameras. In *Proceedings of the Winter Conference on Applications of Computer Vision (WACV)*, pages 5375–5385, January 2023.

[153] Wolfgang Maass. Networks of spiking neurons: The third generation of neural network models. *Neural Networks*, 10(9):1659–1671, December 1997.

[154] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018.

[155] Matteo Maggioni, Vladimir Katkovnik, Karen Egiazarian, and Alessandro Foi. Nonlocal transform-domain filter for volumetric data denoising and reconstruction. *Transactions on Image Processing (TIP)*, 22(1):119–133, 2012.

[156] Dmitrii Marin, Jen-Hao Rick Chang, Anurag Ranjan, Anish Prabhu, Mohammad Rastegari, and Oncel Tuzel. Token pooling in vision transformers. arXiv, 2021.

[157] Lingchen Meng, Hengduo Li, Bor-Chun Chen, Shiyi Lan, Zuxuan Wu, Yu-Gang Jiang, and Ser-Nam Lim. AdaViT: Adaptive vision transformers for efficient image recognition. In *Proceedings of the Conference on*

Computer Vision and Pattern Recognition (CVPR), pages 12309–12318, June 2022.

[158] Yue Meng, Chung-Ching Lin, Rameswar Panda, Prasanna Sattigeri, Leonid Karlinsky, Aude Oliva, Kate Saenko, and Rogerio Feris. AR-Net: Adaptive frame resolution for efficient action recognition. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 86–104, 2020.

[159] Yue Meng, Rameswar Panda, Chung-Ching Lin, Prasanna Sattigeri, Leonid Karlinsky, Kate Saenko, Aude Oliva, and Rogerio Feris. Adafuse: Adaptive temporal fusion network for efficient action recognition. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

[160] Nico Messikommer, Daniel Gehrig, Antonio Loquercio, and Davide Scaramuzza. Event-based asynchronous sparse convolutional networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 415–431, 2020.

[161] Sparsh Mittal. A survey of techniques for approximate computing. *ACM Computing Surveys*, 48(4), March 2016.

[162] Kazuhiro Morimoto, Andrei Ardelean, Ming-Lo Wu, Arin Can Ulku, Ivan Michel Antolovic, Claudio Bruschini, and Edoardo Charbon. Megapixel time-gated SPAD image sensor for 2D and 3D imaging applications. *Optica*, 7(4):346–354, April 2020.

[163] Norman Mu and Justin Gilmer. MNIST-C: A robustness benchmark for computer vision. arXiv, 2019.

[164] Gottfried Munda, Christian Reinbacher, and Thomas Pock. Real-time intensity-image reconstruction for event cameras using manifold regu-

larisation. *International Journal of Computer Vision (IJCV)*, 126:1381–1393, 2018.

[165] Shuichi Namiki, Shunichi Sato, Yusuke Kameda, and Takayuki Hamamoto. Imaging method using multi-threshold pattern for photon detection of quanta image sensor. In *International Workshop on Advanced Imaging Technology (IWAIT)*, volume 12177, page 1217702, 2022.

[166] Daniel Neil, Jun Haeng Lee, Tobi Delbruck, and Shih-Chii Liu. Delta networks for optimized recurrent network computation. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 2584–2593, July 2017.

[167] Xuecheng Nie, Yuncheng Li, Linjie Luo, Ning Zhang, and Jiashi Feng. Dynamic kernel distillation for efficient pose estimation in videos. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6942–6950, 2019.

[168] Peter O'Connor and Max Welling. Sigma delta quantized networks. arXiv, 2016.

[169] Bowen Pan, Wuwei Lin, Xiaolin Fang, Chaoqin Huang, Bolei Zhou, and Cewu Lu. Recurrent residual module for fast inference in videos. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1536–1545, 2018.

[170] Bowen Pan, Rameswar Panda, Yifan Jiang, Zhangyang Wang, Rogerio Feris, and Aude Oliva. IA-RED$\hat{2}$: Interpretability-aware redundancy reduction for vision transformers. In *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, volume 34, pages 24898–24911, 2021.

[171] Liyuan Pan, Richard Hartley, Cedric Scheerlinck, Miaomiao Liu, Xin Yu, and Yuchao Dai. High frame rate video reconstruction based on an event camera. *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 44(5):2519–2533, 2022.

[172] Liyuan Pan, Cedric Scheerlinck, Xin Yu, Richard Hartley, Miaomiao Liu, and Yuchao Dai. Bringing a blurry frame alive at high frame-rate with an event camera. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[173] Zizheng Pan, Bohan Zhuang, Jing Liu, Haoyu He, and Jianfei Cai. Scalable vision transformers with hierarchical pooling. In *Proceedings of the International Conference on Computer Vision (ICCV)*, pages 377–386, October 2021.

[174] Tianyu Pang, Kun Xu, Yinpeng Dong, Chao Du, Ning Chen, and Jun Zhu. Rethinking softmax cross-entropy loss for adversarial robustness. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2020.

[175] Federico Paredes-Valles and Guido C. H. E. de Croon. Back to event basics: Self-supervised learning of image reconstruction for event cameras via photometric constancy. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3446–3455, June 2021.

[176] Mathias Parger, Chengcheng Tang, Christopher D. Twigg, Cem Keskin, Robert Wang, and Markus Steinberger. DeltaCNN: End-to-end CNN inference of sparse frame differences in videos. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12497–12506, June 2022.

[177] Sylvain Paris, Samuel W. Hasinoff, and Jan Kautz. Local Laplacian filters: edge-aware image processing with a Laplacian pyramid. In *SIGGRAPH*, 2011.

[178] Sylvain Paris, Samuel W. Hasinoff, and Jan Kautz. Local Laplacian Filters: Edge-aware image processing with a Laplacian pyramid. *Transactions on Graphics (TOG)*, page 11, 2011.

[179] Vaishakh Patil, Wouter Van Gansbeke, Dengxin Dai, and Luc Van Gool. Don't forget the past: Recurrent depth estimation from monocular video. *Robotics and Automation Letters*, 5(4):6813–6820, 2020.

[180] Federico Perazzi, Anna Khoreva, Rodrigo Benenson, Bernt Schiele, and Alexander Sorkine-Hornung. Learning video object segmentation from static images. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[181] Federico Perazzi, Jordi Pont-Tuset, Brian McWilliams, Luc Van Gool, Markus Gross, and Alexander Sorkine-Hornung. A benchmark dataset and evaluation methodology for video object segmentation. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[182] Christoph Posch, Daniel Matolin, and Rainer Wohlgenannt. A QVGA 143 dB dynamic range frame-free PWM image sensor with lossless pixel-level video compression and time-domain CDS. *IEEE Journal of Solid-State Circuits*, 46(1):259–275, 2011.

[183] Gilles Puy and Patrick Perez. A flexible convolutional solver for fast style transfers. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[184] René Ranftl, Katrin Lasinger, David Hafner, Konrad Schindler, and Vladlen Koltun. Towards robust monocular depth estimation: Mixing

datasets for zero-shot cross-dataset transfer. *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 44(3):1623–1637, 2022.

[185] Yongming Rao, Wenliang Zhao, Benlin Liu, Jiwen Lu, Jie Zhou, and Cho-Jui Hsieh. DynamicViT: Efficient vision transformers with dynamic token sparsification. In *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, volume 34, pages 13937–13949, 2021.

[186] Ramesh Raskar, Amit Agrawal, and Jack Tumblin. Coded exposure photography: motion deblurring using fluttered shutter. In *SIGGRAPH*, pages 795–804, 2006.

[187] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. XNOR-Net: ImageNet classification using binary convolutional neural networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 525–542, 2016.

[188] Nikhila Ravi, Valentin Gabeur, Yuan-Ting Hu, Ronghang Hu, Chaitanya Ryali, Tengyu Ma, Haitham Khedr, Roman Rädle, Chloe Rolland, Laura Gustafson, Eric Mintun, Junting Pan, Kalyan Vasudev Alwala, Nicolas Carion, Chao-Yuan Wu, Ross Girshick, Piotr Dollár, and Christoph Feichtenhofer. SAM 2: Segment anything in images and videos. arXiv, 2024.

[189] Henri Rebecq, René Ranftl, Vladlen Koltun, and Davide Scaramuzza. High speed and high dynamic range video with an event camera. *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2019.

[190] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[191] Hongyu Ren, Hanjun Dai, Zihang Dai, Mengjiao Yang, Jure Leskovec, Dale Schuurmans, and Bo Dai. Combiner: Full attention transformer with sparse computation cost. In *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, volume 34, pages 22470–22482, 2021.

[192] Stephan Richter, Zeeshan Hayder, and Vladlen Koltun. Playing for benchmarks. In *Proceedings of the International Conference on Computer Vision (ICCV)*, pages 2232–2241, October 2017.

[193] S. W. Roberts. Control chart tests based on geometric moving averages. *Technometrics*, 42(1):97–101, 2000.

[194] Alexis Rochas. Single photon avalanche diodes in CMOS technology. Technical report, Citeseer, 2003.

[195] Aurko Roy, Mohammad Saffar, Ashish Vaswani, and David Grangier. Efficient content-based sparse attention with routing transformers. arXiv, 2020.

[196] Manuel Ruder, Alexey Dosovitskiy, and Thomas Brox. Artistic style transfer for videos. In *Proceedings of the German Conference on Pattern Recognition*, pages 26–36, 2016.

[197] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet large scale visual recognition challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, April 2015.

[198] Michael S. Ryoo, A. J. Piergiovanni, Anurag Arnab, Mostafa Dehghani, and Anelia Angelova. TokenLearner: What can 8 learned tokens do for images and videos? arXiv, 2021.

[199] Alberto Sabater, Luis Montesano, and Ana C. Murillo. Robust and efficient post-processing for video object detection. In *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*, pages 10536–10542, October 2020.

[200] Nitin J. Sanket, Chethan M. Parameshwara, Chahat Deep Singh, Ashwin V Kuruttukulam, Cornelia Fermüller, Davide Scaramuzza, and Yiannis Aloimonos. Evdodgenet: Deep dynamic obstacle dodging with event cameras. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, pages 10651–10657, 2020.

[201] Cedric Scheerlinck, Nick Barnes, and Robert Mahony. Continuous-time intensity estimation using event cameras. In *Proceedings of the Asian Conference on Computer Vision (ACCV)*, pages 308–324, 2018.

[202] Cedric Scheerlinck, Henri Rebecq, Daniel Gehrig, Nick Barnes, Robert Mahony, and Davide Scaramuzza. Fast image reconstruction with an event camera. In *Proceedings of the Winter Conference on Applications of Computer Vision (WACV)*, March 2020.

[203] Jenny Schmalfuss, Victor Oei, Lukas Mehl, Madlen Bartsch, Shashank Agnihotri, Margret Keuper, and Andrés Bruhn. RobustSpring: Benchmarking robustness to image corruptions for optical flow, scene flow and stereo. arXiv, 2025.

[204] Trevor Seets, Atul Ingle, Martin Laurenzis, and Andreas Velten. Motion adaptive deblurring with single-photon cameras. In *Proceedings of the Winter Conference on Applications of Computer Vision (WACV)*, pages 1945–1954, January 2021.

[205] Sanchari Sen, Balaraman Ravindran, and Anand Raghunathan. EMPIR: Ensembles of mixed precision deep networks for increased robustness against adversarial attacks. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2020.

[206] Prasan Shedligeri and Kaushik Mitra. Photorealistic image reconstruction from hybrid intensity and event-based sensor. *Journal of Electronic Imaging*, 28(6):063012–063012, 2019.

[207] Prasan Shedligeri, Anupama S., and Kaushik Mitra. A unified framework for compressive video recovery from coded exposure techniques. In *Proceedings of the Winter Conference on Applications of Computer Vision (WACV)*, pages 1600–1609, January 2021.

[208] Mark Sheinin, Yoav Y. Schechner, and Kiriakos N. Kutulakos. Computational imaging on the electric grid. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[209] Evan Shelhamer, Kate Rakelly, Judy Hoffman, and Trevor Darrell. Clockwork convnets for video semantic segmentation. In *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*, volume 9915, pages 852–868, 2016.

[210] Evan Shelhamer, Kate Rakelly, Judy Hoffman, and Trevor Darrell. Clockwork Convnets for video semantic segmentation. In *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*, pages 852–868, 2016.

[211] Sillerkiil. Eesti: Aegviidu siniallikad (Aegviidu blue springs in Estonia), April 2021.

[212] Hyeonjun Sim, Jihyong Oh, and Munchurl Kim. XVFI: eXtreme video frame interpolation. In *Proceedings of the International Conference on Computer Vision (ICCV)*, pages 14489–14498, October 2021.

[213] Timo Stoffregen, Cedric Scheerlinck, Davide Scaramuzza, Tom Drummond, Nick Barnes, Lindsay Kleeman, and Robert Mahony. Reducing the sim-to-real gap for event cameras. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 534–549, 2020.

[214] Binyi Su, Lei Yu, and Wen Yang. Event-based high frame-rate video reconstruction with a novel cycle-event network. In *Proceedings of the International Conference on Image Processing (ICIP)*, pages 86–90, 2020.

[215] Deqing Sun, Xiaodong Yang, Ming-Yu Liu, and Jan Kautz. PWC-Net: CNNs for optical flow using pyramid, warping, and cost volume. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8934–8943, 2018.

[216] Ke Sun, Bin Xiao, Dong Liu, and Jingdong Wang. Deep high-resolution representation learning for human pose estimation. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5693–5703, 2019.

[217] Varun Sundar, Andrei Ardelean, Tristan Swedish, Claudio Bruschini, Edoardo Charbon, and Mohit Gupta. SoDaCam: Software-defined cameras via single-photon imaging. In *Proceedings of the International Conference on Computer Vision (ICCV)*, pages 8165–8176, October 2023.

[218] Varun Sundar, Matthew Dutson, Andrei Ardelean, Claudio Bruschini, Edoardo Charbon, and Mohit Gupta. Generalized event cameras. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 25007–25017, June 2024.

[219] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. arXiv, 2014.

[220] Matias Tassano, Julie Delon, and Thomas Veit. FastDVDnet: Towards real-time deep video denoising without flow estimation. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[221] Zachary Teed and Jia Deng. Raft: Recurrent all-pairs field transforms for optical flow. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 402–419, 2020.

[222] Surat Teerapittayanon, Bradley McDanel, and H. T. Kung. BranchyNet: Fast inference via early exiting from deep neural networks. In *Proceedings of the International Conference on Pattern Recognition (ICPR)*, pages 2464–2469, 2016.

[223] Subarna Tripathi, Zachary C. Lipton, Serge Belongie, and Truong Nguyen. Context matters: Refining object detection in video with recurrent neural networks. arXiv, 2016.

[224] Arin Can Ulku, Claudio Bruschini, Ivan Michel Antolovic, Yung Kuo, Rinat Ankri, Shimon Weiss, Xavier Michalet, and Edoardo Charbon. A 512x512 SPAD image sensor with integrated gating for widefield FLIM. *IEEE Journal of Selected Topics in Quantum Electronics*, 25(1):1–12, January 2019.

[225] Vincent Vanhoucke, Andrew Senior, and Mark Z. Mao. Improving the speed of neural networks on CPUs. In *Conference on Neural Information Processing Systems (NeurIPS) Workshops*, 2011.

[226] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, volume 30, 2017.

[227] Andreas Veit and Serge Belongie. Convolutional networks with adaptive inference graphs. In *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

[228] Ziyu Wan, Bo Zhang, Dongdong Chen, and Jing Liao. Bringing old films back to life. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 17694–17703, June 2022.

[229] Chuan Wang, Haibin Huang, Xiaoguang Han, and Jue Wang. Video inpainting by jointly learning temporal structure and spatial details. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pages 5232–5239, July 2019.

[230] Hao Wang, Weining Wang, and Jing Liu. Temporal memory attention for video semantic segmentation. In *Proceedings of the International Conference on Image Processing (ICIP)*, pages 2254–2258, 2021.

[231] Junke Wang, Xitong Yang, Hengduo Li, Li Liu, Zuxuan Wu, and Yu-Gang Jiang. Efficient video transformers with spatial-temporal token selection. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 69–86, 2022.

[232] Lishun Wang, Miao Cao, and Xin Yuan. EfficientSCI: Densely connected network with space-time factorization for large-scale video snapshot compressive imaging. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 18477–18486, June 2023.

[233] Sinong Wang, Belinda Z. Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. arXiv, 2020.

[234] Wenhai Wang, Enze Xie, Xiang Li, Deng-Ping Fan, Kaitao Song, Ding Liang, Tong Lu, Ping Luo, and Ling Shao. Pyramid vision transformer: A versatile backbone for dense prediction without convolutions. In *Proceedings of the International Conference on Computer Vision (ICCV)*, pages 568–578, October 2021.

[235] Wenjing Wang, Shuai Yang, Jizheng Xu, and Jiaying Liu. Consistent video style transfer via relaxation and regularization. *Transactions on Image Processing (TIP)*, 29:9125–9139, 2020.

[236] Xin Wang, Fisher Yu, Zi-Yi Dou, Trevor Darrell, and Joseph E. Gonzalez. SkipNet: Learning dynamic routing in convolutional networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

[237] Yiran Wang, Min Shi, Jiaqi Li, Zihao Huang, Zhiguo Cao, Jianming Zhang, Ke Xian, and Guosheng Lin. Neural video depth stabilizer. In *Proceedings of the International Conference on Computer Vision (ICCV)*, pages 9466–9476, October 2023.

[238] Yisen Wang, Difan Zou, Jinfeng Yi, James Bailey, Xingjun Ma, and Quanquan Gu. Improving adversarial robustness requires revisiting misclassified examples. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2020.

[239] Yulin Wang, Rui Huang, Shiji Song, Zeyi Huang, and Gao Huang. Not all images are worth 16x16 words: Dynamic transformers for efficient image recognition. In *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, volume 34, pages 11960–11973, 2021.

[240] Zhaohui Wang, Xiao Lin, Abhinav Mishra, and Ram Sriharsha. Online changepoint detection on a budget. In *International Conference on Data Mining Workshops (ICDMW)*, pages 414–420, 2021.

[241] Colin Ware. *Visual Thinking for Design*. Morgan Kaufmann, first edition, April 2008.

[242] Mian Wei, Sotiris Nousias, Rahul Gulve, David B. Lindell, and Kiriakos N. Kutulakos. Passive ultra-wideband single-photon imaging. In

Proceedings of the International Conference on Computer Vision (ICCV), pages 8135–8146, October 2023.

[243] Mian Wei, Navid Sarhangnejad, Zhengfan Xia, Nikita Gusev, Nikola Katic, Roman Genov, and Kiriakos N. Kutulakos. Coded two-bucket cameras for computer vision. In *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

[244] Eric Wong and J. Zico Kolter. Learning perturbation sets for robust machine learning. arXiv, 2020.

[245] Chao-Yuan Wu, Manzil Zaheer, Hexiang Hu, R. Manmatha, Alexander J. Smola, and Philipp Krähenbühl. Compressed video action recognition. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6026–6035, 2018.

[246] Zuxuan Wu, Caiming Xiong, Yu-Gang Jiang, and Larry S Davis. LiteEval: A coarse-to-fine framework for resource efficient video recognition. In *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, volume 32, 2019.

[247] Zuxuan Wu, Caiming Xiong, Chih-Yao Ma, Richard Socher, and Larry S Davis. AdaFrame: Adaptive frame selection for fast video recognition. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1278–1287, 2019.

[248] Bin Xiao, Haiping Wu, and Yichen Wei. Simple baselines for human pose estimation and tracking. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 466–481, 2018.

[249] Chang Xiao, Peilin Zhong, and Changxi Zheng. Enhancing adversarial defense by k-winners-take-all. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2020.

[250] Fanyi Xiao and Yong Jae Lee. Video object detection with an aligned spatial-temporal memory. In *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

[251] Yunyang Xiong, Zhanpeng Zeng, Rudrasis Chakraborty, Mingxing Tan, Glenn Fung, Yin Li, and Vikas Singh. Nyströmformer: A Nyström-based algorithm for approximating self-attention. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, volume 35, pages 14138–14148, May 2021.

[252] Kai Xu, Longyin Wen, Guorong Li, Honggang Qi, Liefeng Bo, and Qingming Huang. Learning self-supervised space-time cnn for fast video style transfer. *Transactions on Image Processing (TIP)*, 30:2501–2512, 2021.

[253] Ran Xu, Fangzhou Mu, Jayoung Lee, Preeti Mukherjee, Somali Chaterji, Saurabh Bagchi, and Yin Li. SmartAdapt: Multi-branch object detection framework for videos on mobiles. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2528–2538, June 2022.

[254] Ran Xu, Chen-lin Zhang, Pengcheng Wang, Jayoung Lee, Subrata Mitra, Somali Chaterji, Yin Li, and Saurabh Bagchi. ApproxDet: Content and contention-aware approximate object detection for mobiles. In *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*, pages 449–462, November 2020.

[255] Rui Xu, Xiaoxiao Li, Bolei Zhou, and Chen Change Loy. Deep flow-guided video inpainting. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[256] Chengshuai Yang, Shiyu Zhang, and Xin Yuan. Ensemble learning priors driven deep unfolding for scalable video snapshot compressive

imaging. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2022.

[257] Le Yang, Yizeng Han, Xi Chen, Shiji Song, Jifeng Dai, and Gao Huang. Resolution adaptive networks for efficient inference. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2369–2378, 2020.

[258] Lihe Yang, Bingyi Kang, Zilong Huang, Zhen Zhao, Xiaogang Xu, Jiashi Feng, and Hengshuang Zhao. Depth Anything V2. In *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, volume 37, pages 21875–21911, 2024.

[259] Yi Yang and Deva Ramanan. Articulated human detection with flexible mixtures of parts. *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 35(12):2878–2890, December 2013.

[260] Yixin Yang, Jinshan Pan, Zhongzheng Peng, Xiaoyu Du, Zhulin Tao, and Jinhui Tang. BiSTNet: Semantic image prior guided bidirectional temporal feature fusion for deep exemplar-based video colorization. *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 46(8):5612–5624, 2024.

[261] Chun-Han Yao, Chia-Yang Chang, and Shao-Yi Chien. Occlusion-aware video temporal consistency. In *Proceedings of the International Conference on Multimedia*, pages 777–785, 2017.

[262] Hongxu Yin, Arash Vahdat, Jose M. Alvarez, Arun Mallya, Jan Kautz, and Pavlo Molchanov. A-ViT: Adaptive tokens for efficient vision transformer. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10809–10818, June 2022.

[263] Guan Yu. Variance stabilizing transformations of Poisson, binomial and negative binomial distributions. *Statistics & Probability Letters*, 79(14):1621–1629, 2009.

[264] Xin Yuan, Yang Liu, Jinli Suo, Frédo Durand, and Qionghai Dai. Plug-and-play algorithms for video snapshot compressive imaging. *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 44(10):7093–7111, 2022.

[265] Xiaoyu Yue, Shuyang Sun, Zhanghui Kuang, Meng Wei, Philip H.S. Torr, Wayne Zhang, and Dahua Lin. Vision transformer with progressive sampling. In *Proceedings of the International Conference on Computer Vision (ICCV)*, pages 387–396, October 2021.

[266] Yanhong Zeng, Jianlong Fu, and Hongyang Chao. Learning joint spatial-temporal transformations for video inpainting. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 528–543, 2020.

[267] Bo Zhang, Mingming He, Jing Liao, Pedro V. Sander, Lu Yuan, Amine Bermak, and Dong Chen. Deep exemplar-based video colorization. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[268] Fan Zhang, Yu Li, Shaodi You, and Ying Fu. Learning temporal consistency for low light video enhancement from single images. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4967–4976, June 2021.

[269] Haokui Zhang, Chunhua Shen, Ying Li, Yuanzhouhan Cao, Yu Liu, and Youliang Yan. Exploiting temporal consistency for real-time video depth estimation. In *Proceedings of the International Conference on Computer Vision (ICCV)*, October 2019.

[270] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. ShuffleNet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6848–6856, 2018.

[271] Zelin Zhang, Anthony J. Yezzi, and Guillermo Gallego. Formulating event-based image reconstruction as a linear inverse problem with deep regularization using optical flow. *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 45(7):8372–8389, 2023.

[272] Yuzhi Zhao, Lai-Man Po, Kangcheng Liu, Xuehui Wang, Wing-Yin Yu, Pengfei Xian, Yujia Zhang, and Mengyang Liu. SVCNet: Scribble-based video colorization network with temporal aggregation. *Transactions on Image Processing (TIP)*, 32:4443–4458, 2023.

[273] Yuzhi Zhao, Lai-Man Po, Wing-Yin Yu, Yasar Abbas Ur Rehman, Mengyang Liu, Yujia Zhang, and Weifeng Ou. VCGAN: Video colorization with hybrid generative adversarial network. *Transactions on Multimedia*, 25:3017–3032, 2023.

[274] Xizhou Zhu, Jifeng Dai, Lu Yuan, and Yichen Wei. Towards high performance video object detection. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7210–7218, 2018.

[275] Xizhou Zhu, Yujie Wang, Jifeng Dai, Lu Yuan, and Yichen Wei. Flow-guided feature aggregation for video object detection. In *Proceedings of the International Conference on Computer Vision (ICCV)*, pages 408–417, 2017.

[276] Xizhou Zhu, Yuwen Xiong, Jifeng Dai, Lu Yuan, and Yichen Wei. Deep feature flow for video recognition. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2349–2358, 2017.

[277] Alex Zihao Zhu, Nikolay Atanasov, and Kostas Daniilidis. Event-based visual inertial odometry. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[278] Yunhao Zou, Yinqiang Zheng, Tsuyoshi Takatani, and Ying Fu. Learning to reconstruct high speed and high dynamic range videos from events. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2024–2033, June 2021.