NEURAL EVENTS: DISCRETE ASYNCHRONOUS AUTOENCODERS FOR EVENT-BASED VISION

Anonymous authors

Paper under double-blind review

ABSTRACT

Event cameras capture dynamic scenes with exceptional temporal fidelity by representing them as a continuous stream of microsecond resolution *events*. Each individual event, however, only carries minimal semantic value, merely signaling a localized brightness change. To derive meaningful signals, downstream algorithms need to quickly integrate cues from a potentially massive torrent of low-information events. Current architectures, however, are easily overwhelmed, struggling to balance capturing fine-grained temporal dynamics and maintaining a manageable data throughput. This paper proposes a framework to re-tokenize event streams into a small set of highly informative *neural events*, each representing a local spatio-temporal context window with a discrete learnable code. Every time this code flips, a neural event is triggered, yielding a highly compressed data stream. We demonstrate that, across object detection and classification, networks trained on neural events are on par or surpass the performance of state-of-the-art approaches while reducing the event rate by a factor of 2.0.

1 INTRODUCTION

Visual information is naturally sparse in time. While much of the world remains predictable, informative signals arise when changes occur, like when a car appears from behind a corner or a water balloon pops. Event cameras capture these salient moments as a stream of per-pixel brightness change measurements, *i.e.* events Gallego et al. (2020). Each event captures the pixel location, sign, and timestamp of this brightness change with microsecond latency and temporal resolution, and can thus represent characteristics of highly dynamic scenes (e.g., fast relative motion, highly-textured environments, and/or flickering lights). Individually, however, brightness change events are too fine-grained, carrying little to no meaningful information for downstream tasks; instead, tasks require scalable representations aggregated from patterns of events.

Current approaches to representing event patterns face a fundamental tradeoff: synchronous approaches represent these patterns as dense CNN or Transformer-based feature maps Zubic et al. (2024); Peng et al. (2023b); Gehrig & Scaramuzza (2023); Li et al. (2022a); Perot et al. (2020) extracted from aggregated and artificially synchronized volumes of events. These feature maps, however, can no longer capture semantic events in an event-driven way. They lack sparsity and precise timing information, leading to wasteful processing, storage, and increased latency. Asynchronous approaches instead capture these patterns as sparse and precisely timed event-based representations, *i.e.* spike trains in Spiking Neural Networks (SNNs) Cordone et al. (2022); Lee et al. (2020) or spatio-temporal feature graphs in Graph Neural Networks (GNNs) Schaefer et al. (2022); Gehrig & Scaramuzza (2024). Event-based representation learning, however, is difficult to scale, especially at high event rates. While GNNs need to construct memory-inefficient event-graphs, SNNs are bound by slow sequential processing and high memory usage. Thus both SNN and GNN-based representations still yield a lower task performance compared to dense feature maps Hao et al. (2025; 2026).

In this work, we address this gap by introducing a method for representing event patterns as scalable patch-level event representations, *i.e. neural events*. Neural events capture precise timing information and are spatially sparse and asynchronous, while summarizing groups of raw and noisy events into high-dimensional feature vectors. At each patch, a Discrete and Asynchronous Encoder recurrently processes event sequences using a linear transformer, producing a sequence of assignment probabilities to discrete codes (*i.e.* tokens) from a learned codebook. Each time the code assignment flips,

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

a neural event is triggered. By using discrete codes as opposed to analog membrane potentials (*i.e.* in SNNs), our method benefits from a low time and memory complexity during training, yielding a low-rate, low-resolution stream of highly informative neural events. To maximize event rate reduction, we propose an unsupervised pre-training strategy which combines reconstruction and temporal smoothness objectives. Our contributions are summarized as follows:

- We introduce an Asynchronous Discrete Encoder for learning *neural events* from raw event streams. Neural events are semantically rich, spatio-temporally sparse, and characterized by their timestamp, patch location, and feature, occupying a compact and discrete latent space.
- We introduce a unique tokenization mechanism that triggers a neural event whenever its code flips from its previous state. This strategy can be viewed as a generalization of level-based sampling to the simplex containing all possible code assignment probabilities.
- We introduce a pretraining strategy that combines reconstruction and temporal smoothness objectives to stabilize code assignment probabilities. It prevents redundant neural event spiking at decision boundaries, leading to higher event rate reductions and more stable codebooks.

We demonstrate that neural events reduce the event rate by a factor of 2.0, while adding only a single bit of extra information per event. Despite this reduction, we show across the tasks of object detection (Sec. 4.1) and classification (Sec. 4.2) that conventional (synchronous) networks trained on neural events perform on par or better than existing synchronous and asynchronous approaches. Since neural events are highly informative, smaller downstream networks can be used, boosting efficiency. Moreover, our encoder only uses 140 KFLOPS per event, $17\times$ lower than the state-of-the-art Hao et al. (2025). Finally, we show that, when paired with an asynchronous task network like Gehrig & Scaramuzza (2024), neural events can reduce computation while boosting detection performance by 11.0 mAP (Sec. 4.1).

2 Related Works

Asynchronous Event-based Processing. To preserve the microsecond temporal resolution and sparsity, several paradigms process event camera data asynchronously to avoid dense, frame-like representations. Early approaches proposed sparse asynchronous convolutions Cannici et al. (2019); Messikommer et al. (2020) to encode the local neighborhood of events, or leveraged PointNet-like architectures to process the stream directly as a point cloud Sekikawa et al. (2019). However, these local feature extractors often struggle to efficiently aggregate long-range temporal dynamics, and their computational cost scales poorly for dense events in highly textured, dynamic scenes.

More recently, works like Cannici et al. (2020); Santambrogio et al. (2024) proposed using classical recurrent architectures such as LSTMs to encode raw events, alongside standard attention-based architectures Kamal et al. (2023); Hamaguchi et al. (2023). While effective at capturing temporal context, LSTMs suffer from slow sequential inference over long sequences, and standard attention mechanisms face a quadratic memory and computational bottleneck with respect to sequence length, rendering them prohibitive for high-frequency event streams. To address sequence scaling, recent frameworks Hao et al. (2025; 2026) leverage linear-time recurrent models like RWKV Peng et al. (2023a; 2024), which admit fast parallel training algorithms based on parallel pre-fix scans, while allowing fast recursive inference at test time.

Other methods treat the event stream as a dynamic spatio-temporal point cloud, forming a graph by connecting events in local neighborhoods and utilizing Graph Neural Networks (GNNs) Gehrig & Scaramuzza (2024); Schaefer et al. (2022); Li et al. (2021); Dampfhoffer et al. (2025). Despite their high downstream accuracy, GNNs face a critical scalability bottleneck: maintaining and continuously updating complex graph edges for millions of arriving events incurs a massive, unpredictable memory overhead, making them difficult to deploy on memory-constrained hardware.

In contrast to these prior works, our work does not merely map raw events into heavy continuous spaces or graphs. Instead, we leverage linear-time sequence modeling to project events into a *discrete* latent space. This fundamentally solves the bandwidth and memory bottlenecks of prior methods by semantically compressing the stream into compact 7-bit tokens, maintaining the temporal context required for high performance while reducing the downstream computational load.

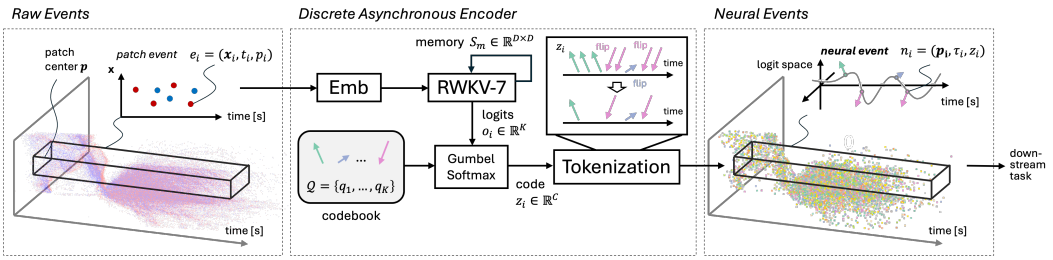


Figure 1: **Method overview.** Our architecture compresses raw events into a low-bandwidth stream of neural events via four stages: (i) Raw events e_i are projected into a continuous vector space using spatial and temporal embeddings. (ii) A linear-attention sequence model (RWKV-7) updates a localized memory state to output a continuous logit vector o_i per event. (iii) A Gumbel Softmax operator maps the continuous logits o_i to a discrete code z_i selected from a pre-learned codebook $\mathcal{Q}$. (iv) Unlike encoding, tokenization acts as an asynchronous, many-to-one semantic compressor. It triggers the proposed neural event n_i only when a code flip occurs, i.e., the current event's code differs from the preceding one ($z_{i_m} \neq z_{i_{m-1}}$). If the code does not flip, the redundant event is discarded, preserving semantic context while minimizing data bandwidth.

Event Rate Reduction and Encoding. As event cameras scale to higher resolutions, the resulting event rates directly inflate storage requirements and saturate real-time transmission bandwidth Gehrig & Scaramuzza (2022). To counteract this data explosion, various lossless Khan et al. (2020) and lossy Banerjee et al. (2021) reduction techniques have been introduced. At the hardware level, modern sensors (e.g., Prophesee Gen4 Finateu et al. (2020)) utilize programmable event-rate controllers and complex read-out schemes to throttle data transmission. These controllers reduce the event rate online by randomly skipping events or dynamically tuning camera biases; however, they permanently discard high-frequency signals and introduce systematic distortions into the event timestamps. Denoising filters sample meaningful events and filter out noises, such as the background activity (BA) filter Lichtsteiner et al. (2008); Delbruck (2008) and other spatio-temporal correlation filters Liu et al. (2015); Guo et al. (2020); Shiba et al. (2025). However, both systematic sampling and denoising remain challenging for practical real-world applications, where it is hard to evaluate these lossy sampling methods. Notably, since they rely on hand-crafted algorithms and parameters, downstream algorithms must be meticulously co-designed with these heuristic filters to account for the degraded signal, inhibiting generalization.

Instead of hand-crafted destructive filtering, learning-based approaches normally convert the event stream to synchronized representations such as images Delbruck (2008); Lagorce et al. (2016) or voxels Zhu et al. (2019). Further data-rate reduction is proposed, leveraging the sparsity of the voxel representation Sezavar et al. (2025). Our approach is an online, learned semantic association by asynchronously re-tokenizing the event stream to neural events. Instead of relying on hardware- or software-level filtering, neural events summarize events in a learned way, bypassing distortions while retaining the critical spatio-temporal dynamics required for downstream perception.

3 Methodology

We first introduce how neural events are defined, and their analogy to standard events in event cameras (Sec. 3.1) before their generation model (Sec. 3.2). We then introduce the *Discrete Asynchronous Encoder* at the heart of neural event generation (Sec. 3.3), and its pretraining (Sec. 3.4).

3.1 Method Overview

Event cameras have independent pixels that record "events" depending on the scene brightness. Let an event stream $\epsilon = (e_1, \ldots, e_N)$ be composed of a sequence of N events $e_i = (\mathbf{x}_i, t_i, p_i)$. Events are triggered whenever the change in log intensity $\log I(\mathbf{x}_i, t_i)$ at pixel $\mathbf{x}_i$ in pixel array Ω, and timestamp t_i exceeds the contrast threshold C Gallego et al. (2020):

$$p_i(\log I(\mathbf{x}_i, t_i) - \log I(\mathbf{x}_i, t_i - \Delta t_i)) = C. \tag{1}$$

Here Δt_i is the time since the last event at the same pixel and p_i is the polarity (sign) of the brightness change. Recent event cameras have high spatial resolutions (i.e., $|\Omega| \sim 10^6$) and thus

generate millions of events per second. In this work, we summarize this high-volume stream of low-information events ϵ, into a low-volume stream of *neural events* ν via mapping ϕ:

$$\phi : \epsilon \mapsto \nu = (n_1, ..., n_{N'}) \quad \text{where} \quad n_i = (\mathbf{p}_i, \tau_i, z_i) \quad \text{and} \quad \mathbf{p}_i \in \Omega', z_i \in \mathcal{Q} \subset \mathbb{R}^C. \tag{2}$$

Each neural event n is characterized by its coordinates $\mathbf{p}$ on a smaller spatial domain Ω', timestamp τ and C-dimensional feature $z_i \in \mathbb{R}^C$. In particular, we design ϕ such that the number of neural events $N' \ll N$, and the domain $|\Omega'| \ll |\Omega|$. Furthermore, as shown later, we design ϕ such that features z_i only take on a *finite set* of K codes in codebook $\mathcal{Q} = \{q_1, q_2, ..., q_K\}$.

3.2 Neural Event Generation Model

Inspired by the working principle of event cameras, we design ϕ to act asynchronously, sparsely, and recursively over spatial memory (see Fig. 1). In particular, we tile the image plane into non-overlapping patches with patch centers $\mathbf{p}_j \in \Omega'$ indexed by j. Let $i_1, .., i_{N_j}$ be the event indices in patch j (a total of N_j). We implement a recursive module with per-patch memory $S_m \in \mathbb{R}^{D\times D}$ indexed by local time index m. Ignoring patch indices for clarity, the update rule is:

$$e'_{i_m}, S_m = \varphi(e_{i_m}, S_{m-1}) \quad \text{with} \quad e'_{i_m} = (\mathbf{p}, t_{i_m}, z_{i_m}). \tag{3}$$

The m^{th} event in the j^{th} patch, e_{i_m}, updates memory S_m at patch j, and emits modified event e'_{i_m}. Note that e'_{i_m} is located at patch center $\mathbf{p}$ and has the same timestamp as the input event t_{i_m} and feature $z_{i_m} \in \mathbb{R}^C$. While the patch-wise design ensures that $|\Omega'| \ll |\Omega|$, the resulting events $\epsilon' = (e'_1, ..., e'_N)$ still retains the same cardinalty N. We thus employ a simple retokenization scheme that only retains events in ϵ' that significantly differ from their predecessor at the same patch:

$$\text{Tokenize} : \epsilon' \mapsto \nu, \quad \text{where} \quad e'_{i_m} \in \nu \iff \|z_{i_m} - z_{\text{ref}}\| > \theta, \tag{4}$$

where z_{ref} denotes the feature of the last neural event in ν triggered at that patch, and θ denotes the neural contrast threshold. As will be shown, restricting the domain of z_{i_m} to a finite codebook and letting $\theta \to 0$ yields an efficient $\mathcal{O}(\max_j \log N_j)$ time complexity implementation of Eqs. (4),(3) during training, and an $\mathcal{O}(1)$ implementation during testing. This enables more scalable training than classical SNNs, which require $\mathcal{O}(\max_j N_j)$ linear time scans.

3.3 Discrete Asynchronous Encoder

Neural event generation in a single patch follows four steps: *(i)* input projection (Emb), followed by *(ii)* recursive application of linear transformer RWKV-7 Peng et al. (2023a), *(iii)* subsequent quantization (Quant), and finally *(iv)* retokenization. Thus steps (*i-iii*) are summarized below:

$$z_{i_m} = \text{Quant}(o_{i_m}) \quad \text{where} \quad o_{i_m}, S^j_m = \text{RWKV-7}(\text{Emb}(e_{i_m}), S^j_{m-1}). \tag{5}$$

Input Embedding. Leveraging the formalism in Fang & Panda (2025), Emb is defined as :

$$\text{Emb}(e_i) = \text{Emb}_s(\mathbf{x}_i, p_i) + \text{Emb}_t(\Delta t_i) \quad \text{where} \quad \text{Emb}_t(\Delta t_i)_k = \begin{cases} \sin\left(\frac{\Delta t_i}{10000^{2k/D}}\right), \text{if k even,} \\ \cos\left(\frac{\Delta t_i}{10000^{2k/D}}\right), \text{if k odd,} \end{cases} \tag{6}$$

where we combine the temporal embedding component Emb_t with the spatial embedding module Emb_s in Fang & Panda (2025) due to its smoothness property with respect to input spatial locations.

Linear Transformer. In what follows we will set $x_{i_m} = \text{Emb}(e_{i_m}) \in \mathbb{R}^D$. Following in the line of linear-time recurrent models Vladymyrov et al. (2024); Gu & Dao (2023), we use a 2-block linear-attention operator RWKV-7 Peng et al. (2025). RWKV-7 mixes temporal features by recursively updating a RWK state S_m at time index m, using receptance $r_{i_m} = W_r x_{i_m}$, key $k_{i_m} = W_k x_{i_m}$, value $v_{i_m} = W_v x_{i_m}$ and temporal decay w_{i_m}

$$S_m = S_{m-1}\left(\text{diag}(w_{i_m}) - \hat{\kappa}_{i_m}(a_{i_m} \odot \hat{\kappa}_{i_m})^\intercal\right) + \tilde{k}_{i_m} v^\intercal_{i_m}. \tag{7}$$

Here, $\hat{\kappa}_{i_m}, a_{i_m} \in \mathbb{R}^D$ are the removal key, and in-context learning-rate. We use the replacement key $\tilde{k}_{i_m}$ derived from the original key k_{i_m}. The output is computed as

$$o_{i_m} = W_o(g_{i_m} \odot p_{i_m}) \quad \text{with} \quad p_{i_m} = \text{LayerNorm}(S_m r_{i_m}) + u_{i_m}, \tag{8}$$

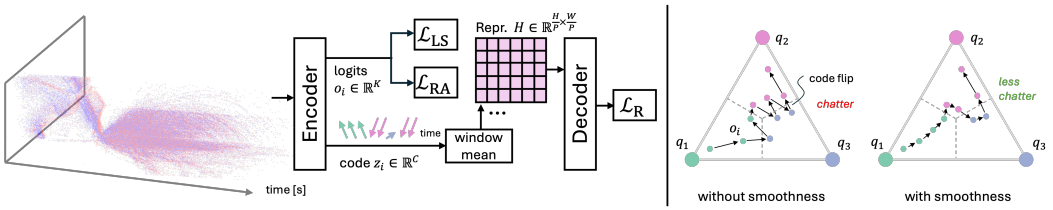


Figure 2: **Pre-training.** Temporally averaged and stacked codes, H are decoded and supervised with time surface Sironi et al. (2018). To mitigate frequent code flipping (chatter), *i.e.* frequent crossing of the decision boundaries (dashed line) in the probability simplex, a rate alignment and latent straightening loss are applied to the logit sequence o_i. This yields smooth trajectories and reduces flipping.

where $u_{i_m} \in \mathbb{R}^D$ is a learned bonus vector that strictly prioritizes the current token, and g_{i_m} is the RWKV gate. Parameters $W_k, W_v, W_r \in \mathbb{R}^{D \times D}$ as well as $W_o \in \mathbb{R}^{K \times D}$ are learnable.

Processing a new event requires exactly $\mathcal{O}(D^2)$ operations and memory, eliminating the need for quadratic attention and KV-caching in standard Transformers. During training, the full sequence of events within a temporal patch is known a priori. Memory S_m in Equation 7 can be efficiently computed over all $k < m$, using a parallel prefix scan Martin & Cundy (2018) with $\mathcal{O}(\max_j N_j \cdot D^2)$ complexity, where N_j denotes event sequence length on patch j. As observed by Wang & Xue (2023) and Li et al. (2022b), memory in linear-recurrent models asymptotically decays exponentially with the number of inputs. Since our transformer operates patch-wise, this decay is different for each patch, and mitigates the forgetting problem in global event encoding in prior approaches Hao et al. (2026). This, in practice, translates to the model struggling to consistently process events from far-away zones of the event image plane. Our patch-wise design instead constrains attention to the local patches, allowing for longer time modeling and approximate shift invariance.

Quantization. Encoding each single event yields continuous-valued vectors $o_i \in \mathbb{R}^D$, which increases the information carried by each single event but doesn't provide any temporal, spatial, or memory compression. To compress our event stream, we seek to re-tokenize events in a more tractable representation. We thus map o_i to a finite set of codes within codebook $Q \in \mathbb{R}^{C \times K}$, formed out of the elements of $\mathcal{Q}$. As shown in Appendix 6.1, this allows neural events to only carry 6% more data than raw events.

We adopt the discrete variational autoencoder formulation Ramesh et al. (2021), which interprets o_i as logits over the columns of Q, and perform straight-through Gumbel softmax assignment via

$$z_i = Qy_i \quad \text{where} \quad y_i = \text{GumbelSoftmax}_\tau(o_i). \tag{9}$$

During the forward pass, y_i behaves as a one-hot vector, and during the backward pass, it behaves as a noise-perturbed softmax with temperature τ. This straight-through assignment is analogous to the spiking mechanism in SNNs Lee et al. (2020), handling discontinuities during training.

Event Retokenization. Here we develop the event trigger condition in Equation (4). In particular, since the sequence $z_{i_1}, z_{i_2}, \ldots, z_{i_{N_j}}$ from sequence $e'_1, \ldots, e'_N$ is piecewise constant, the tokenization scheme can be simplified to

$$\text{Tokenize} : \epsilon' \mapsto \nu \quad \text{where} \quad e'_{i_m} \in \nu \iff z_{i_m} \neq z_{i_{m-1}}, \tag{10}$$

assuming $\theta = 0$ and following a quick derivation in Appendix 6.2. This rule can be interpreted as skipping events whenever there are duplicates, *i.e.* redundant information.

While retaining equivalence to Eq. (4), this rule can be implemented with $\mathcal{O}(1)$ time complexity, unlike the standard spiking rule in SNNs, which requires a hysteresis of arbitrary length, and thus needs $\mathcal{O}(\max_j N_j)$ operations. Moreover, unlike Øhrstrøm et al. (2025), which sets a predefined and explicit refractory period, our neural event triggering depends on the scene dynamics and is learned.

Our tokenization strategy can be seen as a generalization of level-based sampling to the simplex. Let $\Delta^K := \{x \in \mathbb{R}^K | \sum_k x_k = 1\}$ be the simplex, and $p_i := \text{Softmax}(o_i) \in \Delta^K$ denote the probability distribution induced by logits o_i. Then, let $\delta_k := \{x \in \Delta^K | \arg\max x = k\}$. Neural events are thus triggered whenever p_i moves between different δ_k^K. We visualize these regions, together with

distributions p_i over time in Fig. 2. Clearly, shaping how these embeddings evolve temporally can have a large impact on the neural event rate. We will discuss next how this representation is learned.

3.4 Representation Learning

We design representations z_i to (i) be sufficient to perform useful tasks with downstream networks, and (ii) exhibit high compression. We enforce this by pre-training our discrete autoencoder with the following reconstruction and smoothness objectives (see Fig. 2).

$$\mathcal{L} = \mathcal{L}_{\text{R}}(\epsilon, \epsilon') + \omega_2 \mathcal{L}_{\text{RA}}(\epsilon, \epsilon') + \omega_3 \mathcal{L}_{\text{LS}}(\epsilon, \epsilon'). \tag{11}$$

Reconstruction Loss. Our reconstruction loss uses preprocessed events ϵ'_t in Sec.3.3 and transforms them into a synchronous representation H following the codebook voting strategy, depicted in Fig.1. Let n_j^k count the number of events e'_i in patch j associated with code c_k. We form a set of patch embeddings $h_j = \frac{1}{N_j} \sum_{k=1}^{K} n_j^k c_k$ averaging codes according to their frequency in each patch, and concatenate them to an image H. We supervise the decoded image with time-surface $\mathcal{T}(\epsilon)$

$$\mathcal{L}_{\text{R}}(\epsilon, \epsilon') = \omega_0 \|\mathcal{T}(\epsilon) - D_\phi(H)\|_2^2 + \omega_1 \|\mathcal{T}(\epsilon) - D_\phi(H)\|_1. \tag{12}$$

While this objective forces neural event representation to be informative, they are not necessarily smooth, and may therefore lead to frequent code flipping (chatter, see Fig.2). To mitigate this, we employ two temporal smoothing losses, inspired by previous work Wang et al. (2026).

Rate Alignment Loss. As subsequent events with a similar event rate tend to be emitted by the same source in space, they should possess a similar representation. We leverage this intuition to align the logits $(o_{i_m}, o_{i_{m-1}})$ of subsequent events in the same patch, to their event rate r_{i_m}:

$$\mathcal{L}_{\text{RA}} = \sum_j \sum_{m=2}^{N_j} \|\Delta o_{i_m}\|^2 e^{-\gamma |r_{i_m} - r_{i_{m-1}}|} \quad \text{with} \quad \Delta o_{i_m} = o_{i_m} - o_{i_{m-1}}, \tag{13}$$

where $r_{i_m} := \frac{1}{\Delta t_{i_m}}$ and γ is a decay hyperparameter. This loss forces latent representations corresponding to events with small rate differences to be similar.

Latent Straightening Loss. Code assignment when the codebook distribution is uniform is often noisy, generating numerous code flips at the boundaries of code assignment zones. As depicted in Fig. 2, the path of the logits o_i often crosses between two code zones. With temporal straightening Wang et al. (2026), we minimize the velocity angular difference of subsequent logits o_i:

$$\mathcal{L}_{LS} = \sum_j \sum_{m=3}^{N_j} 1 - \frac{\Delta o_{i_m}^\intercal \Delta o_{i_{m-1}}}{\|\Delta o_{i_m}\| \|\Delta o_{i_{m-1}}\|}. \tag{14}$$

4 Experiments

We validate our method on the tasks of object detection (Sec. 4.1), and object classification (Sec. 4.2), showing that our encoder can reduce the original event rate while maintaining highly informative neural events. In Sec. 4.3 we study the impact of codebook size, code dimension, patch size, and pretraining on event rate reduction, computation, and detection performance.

Metrics. Object detection performance is measured in terms of mean Average Precision (mAP), while classification reports top-1 accuracy. Furthermore, we report the MFLOPS/ev and μJ/ev following Gehrig & Scaramuzza (2024). MFLOPS/ev count the computation for each new event; whenever a new event arrives, the synchronous method updates the entire representation, while the asynchronous method only needs to perform local updates. μJ/ev instead measures the computation-induced energy, as measured by Jouppi et al. (2021).

Training details. During pretraining, we reconstruct 2-channel time surfaces from 50-ms event slices from the Gen1 dataset. We set input embedding dimension $D = 64$, codebook size $K = 64$, code dimension $C = 128$, and split the image into patches of 4×5 pixels, resulting in $H/4 \times W/5$ patches. For the small version of the model (TokSwinT-S and TokDAGr-S), we use a 2 RWKV-7 layer encoder,

Table 2: Detection performance on Gen1 and N-Caltech101 datasets. For MFLOPS/ev $*$ represent lower bound from network backbone and N/A undefined due to spike-based computation. For A2S methods we report the representation complexity and in brackets the end-to-end complexity including the decoder.

Method	Async.	Gen1			N-Caltech101		
		mAP↑	MFLOPS/ev↓	μJ/ev↓	mAP↑	MFLOPS/ev↓	μJ/ev↓
Inception+SSD Iacono et al. (2018)	×	30.1	27,183	23,502	-	-	-
Events+RRC Chen (2018)	×	30.7	>21,758*	>18,386	-	-	-
MatrixLSTM+YOLOv3 Cannici et al. (2020)	×	31.0	>34,519*	>29,168	-	-	-
Events+YOLOv3 Jiang et al. (2019)	×	31.2	65,558	55,397	-	-	-
RED Perot et al. (2020)	×	40.0	4,712	3,982	-	-	-
ASTM-Net Li et al. (2022a)	×	46.7	>21,758*	>18,386	-	-	-
HMNet-L3 Hamaguchi et al. (2023)	×	47.1	-	-	-	-	-
RVT-B Gehrig & Scaramuzza (2023)	×	47.2	-	-	-	-	-
GET Peng et al. (2023b)	×	47.9	-	-	-	-	-
NVS-S Simonovsky & Komodakis (2017)	✓	8.60	7.80	6.59	34.6	7.80	6.59
AsyNet Messikommer et al. (2020)	✓	14.5	205	173	64.3	200	169
AEGNN Schaefer et al. (2022)	✓	16.3	5.26	4.44	59.5	7.41	6.26
Spiking DenseNet Cordone et al. (2022)	✓	18.9	N/A	-	-	N/A	-
YOLE Graham et al. (2018)	✓	-	-	-	39.8	3,682	3,111
DAGr-S Gehrig & Scaramuzza (2024)	✓	30.4	4.58	3.87	70.2	6.85	5.76
SSLA-L Hao et al. (2026)	✓	37.5	0.72	0.61	-	-	-
EVA-L Hao et al. (2025)	A2S	47.7	2.32 (3,500)	2.21 (2,958)	-	2.32 (-)	2.21 (-)
Event2Vec Fang & Panda (2025)	A2S	42.5	51.33 (1,181)	43.37 (998)	62.1	51.33 (821)	43.37 (694)
Ours (TokSwinT-S)	A2S	49.9	**0.14** (1,130)	**0.12** (954)	**73.0**	**0.14** (770)	**0.12** (651)
Ours (TokSwinT-L)	A2S	**52.8**	0.62 (1,130)	0.48 (954)	-	-	-

while for the large version (TokSwinT-L), we use 4 RWKV-7 layers. We use eight heads for all three models. The weights of the loss in Eq. 11 are: $\omega_1 = 1.0, \omega_1 = 0.1, \omega_2 = \omega_3 = 0.01$. We pre-train for 100 epochs with a learning rate of 10^{-4}, batch size of 256, and Adam optimizer Kingma & Ba (2014). We then freeze our RWKV-7 encoder and codebook (fixing tokenization), while fine-tuning a task decoder (Swin-T) with the YOLOX detection heads Ge et al. (2021) for additional 100 epochs. We train each step using 64 NVIDIA GH200 GPUs reaching convergence in about 2 hours.

Datasets. We report object detection performance on DSEC-Detection Gehrig et al. (2021), Gen1 De Tournemire et al. (2020), and N-Caltech101 Orchard et al. (2015). DSEC and Gen1 are automotive datasets with ego-centric event cameras capturing a variety of high-speed and stop-and-go motions, while N-Caltech101 Orchard et al. (2015) comprises short sequences recorded on a pan-tilt device observing image projections on a wall, with associated class labels and bounding boxes. We report classification results on N-Caltech101. For more details, see Appendix 6.7.

Baselines. For object detection on DSEC-Detection, we compare our approach to the following three synchronous baselines from Gehrig & Scaramuzza (2024): "Inception+SSD" Iacono et al. (2018), "Events+YOLOv3" Jiang et al. (2019), and "Events+YOLOX" Ge et al. (2021), which combine images and events. On Gen1 and N-Caltech101, we report a variety of synchronous, asynchronous, and asynchronous to synchronous (A2S) methods like Hao et al. (2025). A2S methods compute representations event-by-event, while running the backbone synchronously. Since Event2Vec Fang & Panda (2025) was originally evaluated only on classification tasks, we adapt it for object detection by feeding representations from its publicly available model into our Swin-T decoder. We keep the Event2Vec weights frozen and train only the decoder on Gen1.

4.1 Object Detection

Table 1: Detection performance on DSEC-Detection (mAP), computation (MFLOPS/ev), and energy (μJ/ev).

Model	Images	Async	mAP	MFLOPS/ev	μJ/ev
Inception+SSD Iacono et al. (2018)	✓	×	18.4	27,183	23,501
Events+YOLOv3 Jiang et al. (2019)	✓	×	28.7	65,558	55,396
Events+YOLOX Ge et al. (2021)	✓	×	**40.2**	22,049	18,631
DAGr-S Gehrig & Scaramuzza (2024)	×	✓	14.0	6.05	5.12
Ours (TokDAGr-S)	×	✓	**25.0**	**5.81**	**4.91**

Setup. We pair our tokenizer with both *synchronous* and *asynchronous* backbones. The synchronous backbone chains an input projection to 96 dimensions (starting at representation H), followed by a Swin-T backbone Liu et al. (2022) and a YOLOX task head Ge et al. (2021) as introduced by EVA Hao et al. (2025). The asynchronous backbone consists of a state-of-the-art event-by-event processing graph neural network, DAGr Gehrig & Scaramuzza (2024). For DAGr, we process the input neural events as a reduced spatio-temporal graph and adapt the input dimension to C, the code dimension. We term these TokSwinT and TokDAGr.

Results on DSEC. Tab.1 shows that TokDAGr increases the performance of the base model DAGr by 11.0 mAP, while reducing computation and energy. It thus approaches Events+YOLOv3 without using images, while using roughly 10k times less computation per event. We argue that the improvement over DAGr comes from its ability to encode temporal histories, which enhance detection, especially in stop-and-go settings. This result shows that neural events are simultaneously highly expressive and capable of reducing computation through event-rate reduction.

Results on Gen1 and N-Caltech101. Tab.2 shows the results of TokSwinT. For the A2S models, we separately report the computation and energy required for building the representations and running the entire network (in brackets). TokSwinT-S outperforms synchronous, asynchronous, and A2S models, surpassing the state-of-the-art GET Peng et al. (2023b) and EVA-L Hao et al. (2025). In particular, TokSwinT-S is more efficient around $17\times$ (ours 0.14 and EVA 2.32 MFLOPS/ev) at representation building than EVA-L by, as our RWKV-7 uses fewer layers (2 instead of 3) and a smaller hidden dimension ($D = 64$) than RVT Gehrig & Scaramuzza (2023) ($D = 128$) and EVA-L ($D = 192$).

On N-Caltech101, TokSwin-T asynchronous baselines like Gehrig & Scaramuzza (2024); Schaefer et al. (2022) use only 140 $\times 10^3$ KFLOPS/ev to build the representation. The end-to-end computation, in brackets, is significantly higher based on our asynchronous FLOPS measurement protocol, which is required to run a full forward pass for the Swin-T backbone for every tokenized event.

4.2 Object Classification

Setup. After pretraining for reconstruction and detection on the Gen1 dataset, we finetune our model on N-Caltech101, following the test-split in Gehrig et al. (2019), and report classification scores in Tab.3. We report results for various Asynchronous (A), Synchronous (S), and Asynchronous-to-Synchronous (A2S) baselines.

Table 3: Object classification results on N-Caltech101.

Model	Async.	Acc.
HOTS (Lagorce et al., 2016)	✓	21.0%
NVS-S (Li et al., 2021)	✓	67.0%
AEGNN (Schaefer et al., 2022)	✓	66.8%
FARSE-CNN (Santambrogio et al., 2024)	✓	68.7%
Asynet (Messikommer et al., 2020)	✓	74.5%
EST (Gehrig et al., 2019)	×	83.7%
Matrix-LSTM (Cannici et al., 2020)	×	84.3%
Pillar (Fan et al., 2025)	×	85.3%
E2VID (Rebecq et al., 2019)	×	**86.6%**
EVA-L (Hao et al., 2025)	A2S	86.3%
Ours	A2S	86.5%

Results. Tab 3 shows our method performs on par with several asynchronous Santambrogio et al. (2024); Messikommer et al. (2020), synchronous Rebecq et al. (2019) and A2S Hao et al. (2025) baselines; notably, it performs similarly to E2VID, which requires reconstructing images from events with additional latency. It also performs similarly to EVA, which requires more MFLOPS/ev to compute the representation (2.32) compared to ours (0.14), as shown in Tab 2.

4.3 Ablations and Sensitivity Studies

Setup. We study in Tab. 4 the impact of **(a)** the loss functions in Equation 11, **(b)** the patch size and we isolate the contribution of Neural Event tokenization by comparing in **(c)** against various rate-matched event tokenization methods. We report the object detection performance on the Gen1 dataset (mAP), the resulting event rate reduction ($R := N/N'$) in going from raw to neural events, the computational complexity of the encoder, in terms of MFLOPS/ev and the end-to-end latency in ms to process a packet of 100,000 events. For (a), we re-train the different models from scratch on the same split of the Gen1 training set for 40 epochs, and then fine tune the object detector. For (b) we train the object detector and representation jointly from scratch. We vary the patch size from 2×3 to 8×10, padding the original dimension in the case of the model with a 5×6 patch dimension. For (c) we retrain each of the variant, replicating the same base training schema. We initially train two models using random event dropping and background-activity filtering, removing entirely the neural events tokenizer and leaving the original decoder. We then train three variants of the base model: the first randomly dropping neural events, the second averaging subsequent neural events, instead of using the introduced code flipping pruning schema in Eq.10 and the last one correspond to our base model. Finally we compare two models, using both the full event sequence, without compression. The first uses the raw logits of our linear-attention layer and one using the output of our tokenizer but without code flipping pruning.

Ablation on loss functions. Tab 4a shows that combining both RA and LS losses achieves the highest mAP of 45.8 and a compression rate of $R = 2.0$. Removing RA reduces mAP by 12.1 points, while removing LS further reduces mAP by 3.0. Surprisingly, we find that higher event rate reductions indicate higher task performances. We argue that LS and RA are vital to stabilizing the codebook,

(a) Smoothness losses.

LS	RA	mAP ↑	R ↑
×	×	30.7	1.3
✓	×	33.7	1.6
✓	✓	45.8	2.0

(b) Patch size.

Patch size	mAP ↑	ms ↓
2×3	34.8	37.9
3×4	35.1	37.8
4×5	**43.9**	36.7
5×6	39.8	47.3
8×10	40.0	43.2

(c) Representation and emission.

Representation		Reduction				Results	
RWKV	VQ	Rand.	BA	Avg.	CC	N'/N	mAP ↑
×	×	✓	×	×	×	0.50	25.5
×	×	×	✓	×	×	0.50	29.5
✓	✓	✓	×	×	×	0.50	38.3
✓	✓	×	×	✓	×	0.50	43.7
✓	✓	×	×	×	✓	0.50	**49.9**
✓	×	×	×	×	×	1.00	39.5
✓	✓	×	×	×	×	1.00	**51.1**

Table 4: Ablations on Gen1: (a) smoothness losses, (b) patch size, and (c) representation and emission. The underlined patch size is our default. All variants in (c) use a Swin-T backbone and a YOLOX head. RWKV denotes the learned encoder and VQ vector quantization. Rand., BA, Avg., and CC denote random dropping, background-activity filtering, pairwise code-feature averaging, and code-change emission corresponding to our base model, respectively. Rows without RWKV use filtered event images; averaging is applied after quantization.

as shown in Fig. 3. In each plot we show the empirical log likelihood of transitioning from code $i = 1, ..., 64$ to code $j = 1, ..., 64$. While (b-c) show uniform transition probabilities, indicating code collapse, (a) shows a highly non-uniform distribution. Moreover, (a) shows a higher weight on the diagonal, suggesting that RWKV-7 can preserve diversity by selecting many different codes and semantic association by aggregating different events into the same code, avoiding flip where code $i = j$. Finally, combining RA and LS losses reduces dead codes (dark bands).

Sensitivity to patch size:

Tab. 4b identifies 4×5 as the best tested configuration, with 43.9 mAP and a full-pipeline execution time of 36.7 ms. Larger patches worsen both metrics despite reducing the patch count. For 5×6, padding required to align outputs with Gen1 ground-truth annotations adds computational overhead

Representation and emission: Tab. 4c highlights the contributions of representation learning and selective emission. At $N'/N = 0.5$, random dropping and background-activity filtering of raw events removes our tokenization strategy, retaining instead our task decoder. They achieve 25.5 and 29.5 mAP, respectively, compared with 49.9 for our model, which highlights Neural Event representation space as fundamental for increasing the task performance. The emission rule also matters once the representation is learned. Our base model using only flipped codes achieves 49.9 mAP, compared with 38.3 for random code dropping and 43.7 for pairwise averaging. This shows that our code flipping strategy is able to maximize information while dropping redundant tokens. On the contrary, random dropping and pairwise averaging grouping do not take into account any representation dynamics. They indiscriminately remove distinct code states while retaining redundant information, thereby lowering the task performance as a consequence. Without output suppression, quantized codes achieve 51.1 mAP versus 39.5 for continuous RWKV logits, supporting the benefit of the learned codebook representation. A plausible explanation is that the codebook provides a useful representational bottleneck: logit variations that leave the selected code unchanged map to the same learned feature vector. This invariance can reduce irrelevant variability caused by noisy or uninformative events presented to the detector.

5 Conclusion

We introduce the Discrete Asynchronous Encoder, a novel method for summarizing sequences of raw low-level events into semantically rich neural events. It maps patch event sequences to sequences of assignment probabilities that select discrete codes within a learned codebook. Each time the code flips, a neural event is emitted. Our encoder is $17\times$ more efficient than state-of-the-art asynchronous event encoders, while producing neural events that are highly informative, and exhibit 2 times lower event rates than the original stream. We show that networks trained with these events are more efficient and perform on par with or better than the state-of-the-art across object classification and detection. Our work opens the door to more general forms of event representation learning, beyond event-based cameras. We believe that following down this path will be key to enabling ultra-low-power deployment on memory-constrained edge and embedded hardware.

AI USE STATEMENT

(This section is **required** and does not count toward the page limit.)

In this work, we used generative AI tools to polish the writing and implement efficient vectorized parts of our method. We have not used generative AI tools to generate synthetic data sets, help develop theoretical models or conceptual frameworks, formulate mathematical claims, provide critical ingredients for proving mathematical claims, assist in the writing of proofs, propose or refine hypotheses, design or provide feedback on research methodology or experiments, assist with translation, clean and reformat dataset, support qualitative and thematic data analysis and interpret results.

Additionally, we used generative AI tools for draft parts of a research paper, summarize or analyse existing literature and propose keywords for a research paper. We have reviewed all AI-assisted work. We checked LLM-generated summaries of the literature and the correctness's of results from vectorized code. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work presents no ethical concerns under the Code of Ethics, as the research does not involve human subjects, sensitive dataset releases, privacy or security risks, harmful applications, or conflicts of interest.

REPRODUCIBILITY STATEMENT

We will release the code for reproducibility upon acceptance.

REFERENCES

Srutarshi Banerjee, Zihao W Wang, Henry H Chopp, Oliver Cossairt, and Aggelos K Katsaggelos. Lossy event compression based on image-derived quad trees and poisson disk sampling. In *2021 IEEE International Conference on Image Processing (ICIP)*, pp. 2154–2158. IEEE, 2021.

Marco Cannici, Marco Ciccone, Andrea Romanoni, and Matteo Matteucci. Asynchronous convolutional networks for object detection in neuromorphic cameras. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pp. 0–0, 2019.

Marco Cannici, Marco Ciccone, Andrea Romanoni, and Matteo Matteucci. A differentiable recurrent surface for asynchronous event-based data. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XX 16*, pp. 136–152. Springer, 2020.

Shantanu Chakrabartty, Chetan Singh Thakur, et al. Neuromorphic computing with aer using time-to-event-margin propagation. *arXiv preprint arXiv:2304.13918*, 2023.

Nicholas FY Chen. Pseudo-labels for supervised learning on dynamic vision sensor data, applied to object detection under ego-motion. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pp. 644–653, 2018.

Loïc Cordone, Benoît Miramond, and Philippe Thierion. Object detection with spiking neural networks on automotive event data. In *2022 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8. IEEE, 2022.

Manon Dampfhoffer, Thomas Mesquida, Damien Joubert, Thomas Dalgaty, Pascal Vivet, and Christoph Posch. Graph neural network combining event stream and periodic aggregation for low-latency event-based vision. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 6909–6918, 2025.

Pierre De Tournemire, Davide Nitti, Etienne Perot, Davide Migliore, and Amos Sironi. A large scale event-based detection dataset for automotive. *arXiv preprint arXiv:2001.08499*, 2020.

Tobi Delbruck. Frame-free dynamic digital vision. In *Proc. Int. Symp. Secure-Life Electron.*, pp. 21–26, 2008. doi: 10.5167/uzh-17620.

Rui Fan, Weidong Hao, Juntao Guan, Lai Rui, Lin Gu, Tong Wu, Fanhong Zeng, and Zhangming Zhu. Eventpillars: Pillar-based efficient representations for event data. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 2861–2869, 2025.

Wei Fang and Priyadarshini Panda. Event2vec: Processing neuromorphic events directly by representations in vector space. *arXiv preprint arXiv:2504.15371*, 2025.

Thomas Finateu, Atsumi Niwa, Daniel Matolin, Koya Tsuchimoto, Andrea Mascheroni, Etienne Reynaud, Pooria Mostafalu, Frederick Brady, Ludovic Chotard, Florian LeGoff, et al. 5.10 a $1280\times$ 720 back-illuminated stacked temporal contrast event-based vision sensor with 4.86 μm pixels, 1.066 geps readout, programmable event-rate controller and compressive data-formatting pipeline. In *2020 IEEE International Solid-State Circuits Conference-(ISSCC)*, pp. 112–114. IEEE, 2020.

Guillermo Gallego, Tobi Delbrück, Garrick Orchard, Chiara Bartolozzi, Brian Taba, Andrea Censi, Stefan Leutenegger, Andrew J Davison, Jörg Conradt, Kostas Daniilidis, et al. Event-based vision: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 44(1):154–180, 2020.

Zheng Ge, Songtao Liu, Feng Wang, Zeming Li, and Jian Sun. Yolox: Exceeding yolo series in 2021. *arXiv preprint arXiv:2107.08430*, 2021.

Daniel Gehrig and Davide Scaramuzza. Are high-resolution event cameras really needed? In *arxiv:2203.14672*, 2022.

Daniel Gehrig and Davide Scaramuzza. Low-latency automotive vision with event cameras. *Nature*, 629(8014):1034–1040, 2024.

Daniel Gehrig, Antonio Loquercio, Konstantinos G Derpanis, and Davide Scaramuzza. End-to-end learning of representations for asynchronous event-based data. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 5633–5643, 2019.

Mathias Gehrig and Davide Scaramuzza. Recurrent vision transformers for object detection with event cameras. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 13884–13893, 2023.

Mathias Gehrig, Willem Aarents, Daniel Gehrig, and Davide Scaramuzza. Dsec: A stereo event camera dataset for driving scenarios. *IEEE Robotics and Automation Letters*, 6(3):4947–4954, 2021.

Benjamin Graham, Martin Engelcke, and Laurens Van Der Maaten. 3d semantic segmentation with submanifold sparse convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 9224–9232, 2018.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

Shasha Guo, Ziyang Kang, Lei Wang, Shiming Li, and Weixia Xu. Hashheat: An O(C) complexity hashing-based filter for dynamic vision sensor. In *25th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 452–457, 2020.

Ryuhei Hamaguchi, Yasutaka Furukawa, Masaki Onishi, and Ken Sakurada. Hierarchical neural memory network for low latency event processing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 22867–22876, 2023.

Haiqing Hao, Nikola Zubić, Weihua He, Zhipeng Sui, Davide Scaramuzza, and Wenhui Wang. Maximizing asynchronicity in event-based neural networks. *arXiv preprint arXiv:2505.11165*, 2025.

Haiqing Hao, Zhipeng Sui, Rong Zou, Zijia Dai, Nikola Zubić, Davide Scaramuzza, and Wenhui Wang. Low-latency event-based object detection with spatially-sparse linear attention. *arXiv preprint arXiv:2603.06228*, 2026.

Massimiliano Iacono, Stefan Weber, Arren Glover, and Chiara Bartolozzi. Towards event-driven object detection with off-the-shelf deep learning. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1–9. IEEE, 2018.

Zhuangyi Jiang, Pengfei Xia, Kai Huang, Walter Stechele, Guang Chen, Zhenshan Bing, and Alois Knoll. Mixed frame-/event-driven fast pedestrian detection. In *2019 International Conference on Robotics and Automation (ICRA)*, pp. 8332–8338. IEEE, 2019.

Norman P Jouppi, Doe Hyun Yoon, Matthew Ashcraft, Mark Gottscho, Thomas B Jablin, George Kurian, James Laudon, Sheng Li, Peter Ma, Xiaoyu Ma, et al. Ten lessons from three generations shaped google's tpuv4i: Industrial product. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, pp. 1–14. IEEE, 2021.

Uday Kamal, Saurabh Dash, and Saibal Mukhopadhyay. Associative memory augmented asynchronous spatiotemporal representation learning for event-based perception. In *The Eleventh International Conference on Learning Representations*, 2023.

Nabeel Khan, Khurram Iqbal, and Maria G Martini. Lossless compression of data from static and mobile dynamic vision sensors-performance and trade-offs. *Ieee Access*, 8:103149–103163, 2020.

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

Xavier Lagorce, Garrick Orchard, Francesco Galluppi, Bertram E Shi, and Ryad B Benosman. Hots: a hierarchy of event-based time-surfaces for pattern recognition. *IEEE transactions on pattern analysis and machine intelligence*, 39(7):1346–1359, 2016.

Chankyu Lee, Adarsh Kumar Kosta, Alex Zihao Zhu, Kenneth Chaney, Kostas Daniilidis, and Kaushik Roy. Spike-flownet: event-based optical flow estimation with energy-efficient hybrid neural networks. In *European conference on computer vision*, pp. 366–382. Springer, 2020.

Jianing Li, Jia Li, Lin Zhu, Xijie Xiang, Tiejun Huang, and Yonghong Tian. Asynchronous spatio-temporal memory network for continuous event-based object detection. *IEEE Transactions on Image Processing*, 31:2975–2987, 2022a.

Yijin Li, Han Zhou, Bangbang Yang, Ye Zhang, Zhaopeng Cui, Hujun Bao, and Guofeng Zhang. Graph-based asynchronous event processing for rapid object recognition. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 934–943, 2021.

Zhong Li, Jiequn Han, Qianxiao Li, et al. Approximation and optimization theory for linear continuous-time recurrent neural networks. *Journal of Machine Learning Research*, 23(42):1–85, 2022b.

Patrick Lichtsteiner, Christoph Posch, and Tobi Delbruck. A 128×128 120 dB 15 μs latency asynchronous temporal contrast vision sensor. *IEEE J. Solid-State Circuits*, 43(2):566–576, 2008. doi: 10.1109/JSSC.2007.914337.

Hongjie Liu, Christian Brandli, Chenghan Li, Shih-Chii Liu, and Tobi Delbruck. Design of a spatiotemporal correlation filter for event-based sensors. In *IEEE Int. Symp. Circuits Syst. (ISCAS)*, pp. 722–725, 2015. doi: 10.1109/ISCAS.2015.7168735.

Ze Liu, Han Hu, Yutong Lin, Zhuliang Yao, Zhenda Xie, Yixuan Wei, Jia Ning, Yue Cao, Zheng Zhang, Li Dong, et al. Swin transformer v2: Scaling up capacity and resolution. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 12009–12019, 2022.

Eric Martin and Chris Cundy. Parallelizing linear recurrent neural nets over sequence length. In *International Conference on Learning Representations*, 2018.

Nico Messikommer, Daniel Gehrig, Antonio Loquercio, and Davide Scaramuzza. Event-based asynchronous sparse convolutional networks. In *European Conference on Computer Vision*, pp. 415–431. Springer, 2020.

Christoffer Koo Øhrstrøm, Ronja Güldenring, and Lazaros Nalpantidis. Spiking patches: Asynchronous, sparse, and efficient tokens for event cameras. *arXiv preprint arXiv:2510.26614*, 2025.

Garrick Orchard, Ajinkya Jayawant, Gregory K Cohen, and Nitish Thakor. Converting static image datasets to spiking neuromorphic datasets using saccades. *Frontiers in neuroscience*, 9:437, 2015.

Bo Peng, Eric Alcaide, Quentin Anthony, Alon Albalak, Samuel Arcadinho, Stella Biderman, Huanqi Cao, Xin Cheng, Michael Chung, Matteo Grella, et al. Rwkv: Reinventing rnns for the transformer era. *arXiv preprint arXiv:2305.13048*, 2023a.

Bo Peng, Daniel Goldstein, Quentin Anthony, Alon Albalak, Eric Alcaide, Stella Biderman, Eugene Cheah, Teddy Ferdinan, Haowen Hou, Przemysław Kazienko, et al. Eagle and finch: Rwkv with matrix-valued states and dynamic recurrence. *arXiv preprint arXiv:2404.05892*, 3, 2024.

Bo Peng, Ruichong Zhang, Daniel Goldstein, Eric Alcaide, Xingjian Du, Haowen Hou, Jiaju Lin, Jiaxing Liu, Janna Lu, William Merrill, et al. Rwkv-7" goose" with expressive dynamic state evolution. In *Second Conference on Language Modeling*, 2025.

Yansong Peng, Yueyi Zhang, Zhiwei Xiong, Xiaoyan Sun, and Feng Wu. Get: Group event transformer for event-based vision. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 6038–6048, 2023b.

Etienne Perot, Pierre De Tournemire, Davide Nitti, Jonathan Masci, and Amos Sironi. Learning to detect objects with a 1 megapixel event camera. *Advances in Neural Information Processing Systems*, 33:16639–16652, 2020.

Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation. In Marina Meila and Tong Zhang (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 8821–8831. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/ramesh21a.html.

Henri Rebecq, René Ranftl, Vladlen Koltun, and Davide Scaramuzza. High speed and high dynamic range video with an event camera. *IEEE transactions on pattern analysis and machine intelligence*, 43(6):1964–1980, 2019.

Riccardo Santambrogio, Marco Cannici, and Matteo Matteucci. Farse-cnn: Fully asynchronous, recurrent and sparse event-based cnn. In *European Conference on Computer Vision*, pp. 1–18. Springer, 2024.

Simon Schaefer, Daniel Gehrig, and Davide Scaramuzza. Aegnn: Asynchronous event-based graph neural networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 12371–12381, 2022.

Yusuke Sekikawa, Kosuke Hara, and Hideo Saito. Eventnet: Asynchronous recursive event processing. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 3887–3896, 2019.

Ahmadreza Sezavar, Catarina Brites, João Ascenso, and Touradj Ebrahimi. A learning-based lossless event data compression for computer vision applications. In *Applications of Digital Image Processing XLVIII*, volume 13605, pp. 230–236. SPIE, 2025.

Shintaro Shiba, Yoshimitsu Aoki, and Guillermo Gallego. Simultaneous motion and noise estimation with event cameras. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 6959–6969, 2025.

Martin Simonovsky and Nikos Komodakis. Dynamic edge-conditioned filters in convolutional neural networks on graphs. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3693–3702, 2017.

Amos Sironi, Manuele Brambilla, Nicolas Bourdis, Xavier Lagorce, and Ryad Benosman. Hats: Histograms of averaged time surfaces for robust event-based object classification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1731–1740, 2018.

Max Vladymyrov, Johannes Von Oswald, Mark Sandler, and Rong Ge. Linear transformers are versatile in-context learners. *Advances in Neural Information Processing Systems*, 37:48784–48809, 2024.

Shida Wang and Beichen Xue. State-space models with layer-wise nonlinearity are universal approximators with exponential decaying memory. *Advances in Neural Information Processing Systems*, 36:74021–74038, 2023.

Ying Wang, Oumayma Bounou, Gaoyue Zhou, Randall Balestriero, Tim GJ Rudner, Yann LeCun, and Mengye Ren. Temporal straightening for latent planning. *arXiv preprint arXiv:2603.12231*, 2026.

Alex Zihao Zhu, Liangzhe Yuan, Kenneth Chaney, and Kostas Daniilidis. Unsupervised event-based learning of optical flow, depth, and egomotion. In *IEEE Conf. Comput. Vis. Pattern Recog. (CVPR)*, pp. 989–997, 2019. doi: 10.1109/CVPR.2019.00108.

Nikola Zubic, Mathias Gehrig, and Davide Scaramuzza. State space models for event cameras. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 5819–5828, 2024.

6 APPENDIX

6.1 MEMORY COMPRESSION

Typically, raw event representations store the $(\mathbf{x}, p, t)$ information, with various precision and optimization strategies. For example, AEDAT 4.0 requires 96 bits (32 bits for $(\mathbf{x}, p)$ and 64 bits for t) Banerjee et al. (2021), and EVT requires 32–58 bits depending on different versions. Alternatively, a common format that does not store the timestamp is AER (Address Event Representation), encoding each event in 19 bits for (x, y, p), however, it is limited in use only for neuromorphic processing Chakrabartty et al. (2023).

The resulting neural events can be represented by their patch coordinate $\mathbf{p}$, timestamp t_{i_m}, and code z_{i_m}. Let b_e be the number of bits used to encode a regular event. The total number of bits to encode a stream of N' neural events is then

$$b_{\text{NE}} \approx N' \left(b_e - \log_2 \frac{|\Omega|}{|\Omega'|} + \log_2 K - 1 \right) \tag{15}$$

where $\log_2 \frac{|\Omega|}{|\Omega'|}$ counts the number of bits saved due to having fewer patches than pixels, and $\log_2 K - 1$ counts the added bits for the code index in $\mathcal{Q}$, instead of a polarity. This yields a compression ratio

$$R = \frac{N'}{N} \frac{b_e - \log_2 \frac{|\Omega|}{|\Omega'|} + b_i + \log_2 K - 1}{b_e} \approx 1.06 \frac{N'}{N} \tag{16}$$

where N counts the number of regular events. We assumed the EVT 3.0 encoding ($b_e = 16$ bits), $\mathcal{Q}$ with a small cardinality $|\mathcal{Q}| = 64$ and a patch size of 4×4, yielding a 6% bit increase per event. However, as will be shown the event rate reduction N'/N compensates for that, yielding $R < 1$.

6.2 DERIVATION OF EQ. (10)

Starting with event generation rule

$$\text{Tokenize} : \epsilon' \mapsto \nu, \quad \text{where} \quad e'_{i_m} \in \nu \iff \|z_{i_m} - z_{\text{ref}}\| > \theta, \tag{17}$$

and letting $\theta \to 0$ yields

$$\text{Tokenize} : \epsilon' \mapsto \nu, \quad \text{where} \quad e'_{i_m} \in \nu \iff z_{i_m} \neq z_{\text{ref}}. \tag{18}$$

Now let $z_{\text{ref}} = z_{i_{m-h}}$ be triggered h time steps before z_{i_m}. Since z_{i_m} is piecewise constant and no events were triggered between i_m and i_{m-h}, we can set $z_{i_{m-h}} = z_{i_{m-1}}$, this yields the simplified $\mathcal{O}(1)$ trigger rule in Eq. (10).

6.3 ABLATIONS ON REPRESENTATION SIZE

We study the effects of the code dimension C and codebook size K on Gen1 detection accuracy and encoder computation. Both sweeps use joint training of the representation and detector from scratch: we vary C with $K = 64$, and vary K with $C = 64$.

Code dimension. Table 5a shows that increasing C from 64 to 128 improves mAP from 40.7 to 43.9, while encoder computation increases only from 0.13 to 0.14 MFLOPs per raw event. Further increasing C to 256 reduces mAP to 34.0 and increases computation to 0.16 MFLOPs per event. Thus, accuracy does not improve monotonically with code dimension, and additional feature capacity at $C = 256$ does not benefit detection under this training protocol. With a shared codebook, neural events transmit code indices, so changing C at fixed K does not change the number of bits per index. We therefore adopt $C = 128$, which achieves the highest accuracy among the tested dimensions with a small computational increase over $C = 64$ and the same code-index width.

Codebook size. Table 5b shows that increasing K from 64 to 128 and 256 improves mAP from 40.7 to 41.8 and 43.3, respectively. However, encoder computation increases from 0.13 to 0.43 and 1.38 MFLOPs per raw event. Across this range, a gain of 2.6 mAP points requires approximately $10.6\times$ more encoder computation. The larger codebooks therefore provide additional accuracy

(a) Code dimension C ($K = 64$).

C	mAP ↑	MFLOPs/ev ↓
64	40.7	0.13
<u>128</u>	43.9	0.14
256	34.0	0.16

(b) Codebook size K ($C = 64$).

K	mAP ↑	MFLOPs/ev ↓
<u>64</u>	40.7	0.13
128	41.8	0.43
256	43.3	1.38

Table 5: Sensitivity to (a) code dimension C and (b) codebook size K on Gen1. Underlined values indicate the default configuration. Computation is reported in MFLOPs per raw event.

(a) NVIDIA GH200.

Component	Time [ms]
Neural Events encoder	10.53
Swin backbone	16.80
YOLOX head + overhead	9.36
Full pipeline	**36.70**

(b) Jetson AGX Orin (64 GB).

Metric	Value
Encoder time	2.67 ms
Encoder rate	3.75×10^6 ev/s
Detector time	37 ms
Detector rate	27 Hz
Encoder memory	281 KB
Detector memory	732.03 MB

Table 6: Hardware measurements on Gen1. (a) GH200 execution times for 100,000 raw events. (b) Jetson component measurements, with encoder windows of 10,000 raw events. Jetson processing rates refer to individual components; the detector rate excludes encoding. Memory values are reported component allocations.

at a substantially higher computational cost in the tested configurations. The codebook size also determines the transmitted index width: with fixed-length coding, each index requires $\lceil \log_2 K \rceil$ bits, corresponding to 6, 7, and 8 bits for $K = 64$, 128, and 256, respectively. Increasing K therefore raises both encoder computation and the number of bits per index. These results motivate our default configuration of $C = 128$ and $K = 64$, which balances detection accuracy and encoder computation while retaining a compact 6-bit code index.

6.4 Execution Time and Hardware Evaluation

We complement the computational-cost analysis with execution-time measurements on an NVIDIA GH200 and an NVIDIA Jetson AGX Orin with 64 GB of memory. Both evaluations use Gen1 at a resolution of 240×304 pixels and the TokSwinT pipeline, comprising the Neural Events encoder, a Swin backbone, and a YOLOX detection head. We distinguish event encoding from downstream detection to identify their respective contributions to runtime.

NVIDIA GH200. Table 6a reports execution times for a packet of 100,000 raw events. Encoding requires $10.53\,\mathrm{ms}$, followed by $16.80\,\mathrm{ms}$ for the Swin backbone and $9.36\,\mathrm{ms}$ for the YOLOX head and associated overhead. The complete pipeline takes $36.70\,\mathrm{ms}$. Encoding therefore accounts for approximately 29% of the measured pipeline runtime, with most execution time spent in downstream detection. This breakdown highlights the contribution of the task network to the overall cost of the asynchronous-to-synchronous pipeline.

NVIDIA Jetson AGX Orin. Table 6b reports component-level measurements on the embedded platform. The encoder processes a window of 10,000 raw events in $2.67\,\mathrm{ms}$, corresponding to an encoder processing rate of approximately 3.75×10^6 raw events per second. The detector requires $37\,\mathrm{ms}$ per inference, corresponding to approximately $27\,\mathrm{Hz}$ for the detector alone.

6.5 Code flipping sensitivity

Figure 3 illustrates how the training objectives affect code-flipping behavior. Each matrix reports empirical log probabilities of transitioning from code i to code j. Diagonal entries ($i = j$) correspond to retaining the current code, whereas off-diagonal transitions ($i \neq j$) trigger new neural events. The diagonal therefore captures the code persistence that enables event-rate reduction.

The full objective in Figure 3(a) produces a more structured transition pattern than reconstruction alone in Figure 3(c), with diagonal activations indicating stable code assignments. This behavior is



(a) LS + RA + R (b) LS + R (c) R

Figure 3: Empirical log probabilities of transitioning from code i to code j for our model trained with (a) reconstruction (R), rate-alignment (RA), and latent straightening losses (LS), (b) LS + R, and (c) R . Dark lines indicate dead codes. Combining RA and LS losses yields highly non-uniform transition distributions with strong diagonal activations ($i = j$), while omitting them causes codebook collapse (right). Note that low off-diagonal components indicate higher event rate reduction.

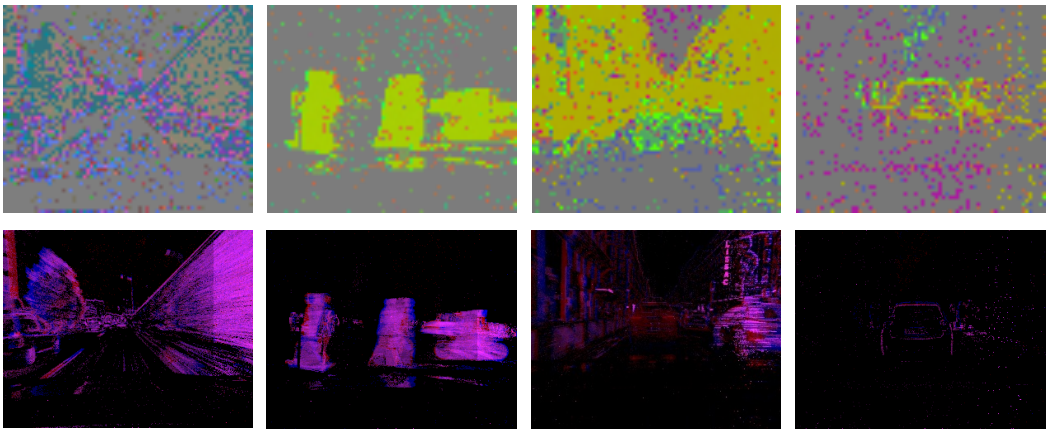


Figure 4: **Spatial grouping in Neural Events.** Top: each patch is colored by its most frequently selected codebook index within the visualized event window. Bottom: corresponding event visualizations. Shared code assignments form coherent regions that follow parts of object silhouettes and scene structures, without an explicit segmentation objective. Colors denote codebook entries, not semantic classes.

consistent with the roles of latent straightening (LS) and rate alignment (RA): LS discourages repeated crossings of code-assignment boundaries, while RA encourages similar logits for consecutive events with similar rates. Dark bands indicate inactive codes, highlighting the importance of assessing codebook usage alongside code persistence.

6.6 Emerging Grouping of Neural Events

Fig. 4 visualizes, for each patch, the most frequently selected codebook index over the displayed event window, assigning a different color to each index. The resulting maps exhibit spatially coherent regions that follow parts of object silhouettes and scene structures, despite the absence of an explicit segmentation objective.

A plausible explanation is the coupling between learned codes and local event dynamics. Event activity depends jointly on local motion and brightness structure; neighboring patches on the same structure can therefore exhibit related temporal activity. The recurrent encoder integrates this history, while the shared codebook allows similar patterns to reuse a discrete prototype. Quantization can thus translate similarities in local event dynamics into common code assignments across spatial locations.

The training objectives may reinforce this organization. Time-surface reconstruction encourages codes to retain the spatial and temporal structure of event activity. Rate alignment favors similar logits for successive events with similar rates, while latent straightening discourages oscillatory

trajectories across code boundaries. Although these regularizers act temporally within each patch, they can stabilize the local states from which spatial grouping emerges. These observations suggest that Neural Events codebook act as a vocabulary of local dynamical patterns.

6.7 Dataset Details

DSEC-Detection. The DSEC-Detection dataset Gehrig et al. (2021) is a high-resolution automotive benchmark recorded using a Prophesee Gen3.1 event camera, which provides a spatial resolution of 640×480 pixels. The dataset comprises 53 sequences (roughly 1 hour of total driving time) captured in and around Zurich, Switzerland. It features a wide distribution of challenging lighting and weather conditions, including night-time, day-time, and high-dynamic-range sunset sequences. The dataset provides ground-truth bounding boxes at a high frequency (up to 20 Hz), which are generated by leveraging synchronized RGB images and LiDAR data. It includes annotations for eight distinct traffic participant classes: *pedestrian*, *rider*, *car*, *bus*, *truck*, *bicycle*, *motorcycle*, and *train*, making it one of the most comprehensive benchmarks for urban event-based object detection.

Gen1 Automotive Detection. The Prophesee Gen1 Automotive dataset De Tournemire et al. (2020) is a large-scale benchmark designed for object detection in realistic driving scenarios. It was recorded using an ATIS-based sensor with a smaller spatial resolution of 304×240 pixels. Despite the lower resolution, the dataset is massive, containing approximately 39 hours of driving data and over 255000 annotated bounding boxes spanning two primary classes: *pedestrians* and *cars*. Gen1 is characterized by a high variance in motion patterns that heavily influence the event generation rate. It features frequent stop-and-go ego-motion (resulting in periods with very few events), a significant component of sensor noise in low-light conditions, and dynamic scenes with traffic participants moving at high relative speeds. This high temporal variance makes it a rigorous testbed for asynchronous and state-based models.

N-Caltech101. Unlike the previous automotive datasets, N-Caltech101 Orchard et al. (2015) (Neuromorphic-Caltech101) is a standard benchmark for event-based object *classification*. It was generated by capturing the original Caltech-101 image dataset using an ATIS event camera with a resolution of 240×180 mounted on a motorized pan-tilt unit. The camera undergoes three distinct, saccade-like triangular motions in front of an LCD monitor displaying the static images. Consequently, the event stream is driven entirely by the artificial motion of the camera over the 2D spatial gradients of the original RGB images. The dataset consists of 8,284 total samples divided into 100 distinct object categories plus a background class. It serves as a fundamental benchmark to evaluate an architecture's ability to extract complex spatial and structural features from event streams generated by consistent, predictable motion.

6.8 Limitations

Despite its computational and bandwidth advantages, our tokenizer uses fixed, non-overlapping spatial patches with independent recurrent states. Information about objects crossing patch boundaries is therefore not explicitly transferred between encoder states, leaving these associations to downstream networks. Future work will explore dynamic spatial patching and context-adaptive codebooks to better capture evolving event patterns. Moreover, tokenization currently occurs after sensor readout, so raw events must still be transmitted to the encoder. Integrating neural-event generation directly into sensor readout hardware could reduce this communication overhead, although its implementation cost and energy benefits remain to be evaluated.