ALERT-Transformer: Bridging Asynchronous and Synchronous Machine Learning for Real-Time Event-based Spatio-Temporal Data

Carmen Martin-Turrero [*1,2] **Maxence Bouvier** [*1] **Manuel Breitenstein** [1] **Pietro Zanuttigh** [2] **Vincent Parret** [1]

Abstract

We seek to enable classic processing of continuous ultra-sparse spatiotemporal data generated by event-based sensors with dense machine learning models. We propose a novel hybrid pipeline composed of asynchronous sensing and synchronous processing that combines several ideas: (1) an embedding based on PointNet models – the ALERT module – that can continuously integrate new and dismiss old events thanks to a leakage mechanism, (2) a flexible readout of the embedded data that allows to feed any downstream model with always up-to-date features at any sampling rate, (3) exploiting the input sparsity in a patch-based approach inspired by Vision Transformer to optimize the efficiency of the method. These embeddings are then processed by a transformer model trained for object and gesture recognition. Using this approach, we achieve performances at the state-of-the-art with a lower latency than competitors. We also demonstrate that our asynchronous model can operate at any desired sampling rate.

1. Introduction

Event-based sensors capture visual information in an event-driven, asynchronous manner (Finateu et al., 2020; Gallego et al., 2020). Efficiently exploiting their data has proven challenging as the vast majority of approaches published in the literature consist of either converting event-based data to dense representations, or deploying spiking neural networks (SNNs) on streams of events. The former allows to exploit standard machine learning (ML) frameworks such as `PyTorch` and `Tensorflow`, but does not leverage the inherent sparsity and other properties of event-based data (Gehrig et al., 2019). The latter relies on SNNs, which are hard to train and usually exhibit lower accuracy than an equivalent dense neural network. Furthermore, while the neuromorphic community has argued in favor of their higher energy efficiency for decades, recent research and breakthroughs in edge AI accelerators indicate this is still an open question (Dampfhoffer et al., 2023; Garrett et al., 2023; Moosmann et al., 2023; Caccavella et al., 2023).

Nevertheless, considering the inherent advantages of event-based vision sensors, namely high dynamic range (HDR) and high temporal resolution – simultaneously, without any tradeoffs between the two –, we aim to find a way to leverage this sparse and low-latency data for real-world situations.

Standard ML relies on tensor-based processing. Converting the stream of events – represented as tuples of values (x and y pixel coordinates, polarities and timestamp) – to a multidimensional tensor is thus a crucial step. The challenge involves (1) representing time in a reliable and continuous manner, allowing it to be processed similarly to the finite spatial and polarity dimensions, (2) continuously incorporating new events in the feature tensors which also requires forgetting previous events, (3) using limited computational resources to allow real-time processing. Our main contributions towards Event-Based ML are the following:

- The ALERT module, an embedding based on PointNet which continuously integrates new events dismissing old ones via a leakage mechanism. This module introduces novel asynchronous embedding updates.
- A flexible readout of the embedded data that can feed any downstream model with up-to-date features at different sampling rates, down to a per-event operation, allowing ultra-low latency decision making.
- A patch-based approach inspired by Vision Transformer to exploit input sparsity and optimize efficiency.
- A time encoding solution to represent continuous time as a bi-dimensional vector of bounded values, at the cost of negligible decrease of relative accuracy.
- The ALERT-Transformer, a framework incorporating all of the above, which is trained on event-based data end-to-end. The model can then operate in synchronous regime for high accuracy on gesture recognition, or asynchronously for ultra-low latency.

[*]Equal contribution [1]Sony Semiconductor Solutions Europe, Sony Europe B.V, Stuttgart Laboratory 1, Zurich, Switzerland [2]University of Padova, MEDIA Lab, Veneto, Italy. Correspondence to: Vincent Parret <vincent.parret@sony.com>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

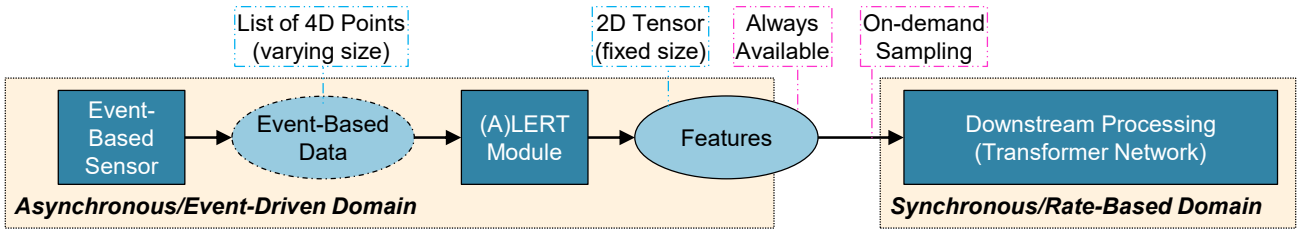

Figure 1. Schematic representation of the system integrating our proposed asynchronous embedding module. The asynchronous part (left) processes all events as they come in an event driven manner, thus updating the $Features$ tensor continuously. The synchronous part (right) samples the $Features$ tensor on demand, thus allowing a seamless interface between asynchronous and synchronous processing.

Our idea is that, even if the sensor outputs data at 1MHz or more (Finateu et al., 2020), the application/client side will not, and does not need to, run at this speed. For instance, in an embedded product, such as a drone, that relies on event-based sensors for self-localization in space, this localization step would not operate at more than 100Hz (Kaufmann et al., 2023). Therefore, we consider that this hybrid (asynchronous-to-synchronous) conversion module would be operated as close to the sensor as possible, inside a 3D-stacked image sensor, as proposed in (Bouvier et al., 2021; Bonazzi et al., 2023).

1.1. Related Works

1.1.1. Event-Based Data Representations

Event-based data is extremely sparse, with unpredictable sparsity patterns, making it unsuitable for raw processing with dense machine learning pipelines. An extended approach consists in integrating events over fixed time windows, creating frame-like representations using methods such as histograms or event queues (Innocenti et al., 2021; Sabater et al., 2022; Maqueda et al., 2018; Zhu et al., 2018). To maximize task accuracy from end-to-end, (Gehrig et al., 2019) proposed learning the kernel for convolving the stream of events into a discrete tensor, called "Event-Spike Tensor". While the resulting networks perform better, this method still requires waiting to accumulate all events before starting processing and is thus not asynchronous.

1.1.2. PointNet Architecture

The PointNet (Qi et al., 2017a;b) is an architecture designed for processing three-dimensional point clouds. It extracts features from individual points with a shared Multi-Layer Perceptron (MLP). It then reduces the entire point cloud with a max-pooling operation into a single global feature vector. Given the similarity in data representation, several works interpreted event streams as point clouds (Wang et al., 2019a; Chen et al., 2022; Zhao et al., 2021; Wang et al., 2019b). These approaches mostly treat time as any other coordinate, making fast and efficient processing challenging.

EventNet (Sekikawa et al., 2019) addresses this issue by processing time separately from other coordinates. They propose a modification to the max operator which becomes a recursive function capable of updating temporal and spatial information as new events arrive. Our solution, introduced in Section 2, eliminates the need to modify the max operator. The structure of the PointNet remains unaltered and is thus entirely compatible with existing AI accelerators.

1.1.3. Event-Based Transformers

Our work takes inspiration from PointBERT (Yu et al., 2022), which combines PointNets with a Transformer (Vaswani et al., 2017) for 3D classification. Models proposed by (Sabater et al., 2022; 2023; Wang et al., 2022; Peng et al., 2023) generate frames of aggregated events using various strategies, and convert them into tokens with a patch-to-token strategy inspired by the Vision Transformer (ViT[1]) (Dosovitskiy et al., 2021). They leverage spatial sparsity by discarding input patches lacking sufficient events. In (Peng et al., 2023), grouped convolutions are used to embed successive patches through time, each token thus handles information from different time steps. (Chen et al., 2022) attempts to input a reduced 3D event cloud to the Point Transformer (Zhao et al., 2021), a model originally designed for spatial 3D point clouds. Following a different approach, (Blegiers et al., 2023) employ a Video Transformer Network using event-frames. All these approaches demonstrate high accuracy, motivating our choice of using Transformer models. However, most are not trained end-to-end, and none can process events asynchronously.

1.1.4. Asynchronous Processing

Besides (Sekikawa et al., 2019), some works focus on asynchronous processing of event streams with standard network architectures. (Sironi et al., 2018) presented an architecture using local memory units shared by neighboring pixels. (Messikommer et al., 2020) introduced Asynchronous Sparse Convolutional Networks, a framework for converting models trained on synchronous image-like event representations into asynchronous models. Even though these

[1] In ViT, images are split into fixed-size patches, which are then linearly embedded to obtain a sequence of vectors.

solutions are built for event-driven processing, they add on complexity with respect to standard ML tools.

2. Proposed Architecture

Our objective is to employ ML techniques to build input embeddings from an asynchronous stream of events, while preserving the properties of event-based data. For this, we combine a PointBERT architecture (Yu et al., 2022) with an embedding inspired by EventNet (Sekikawa et al., 2019).

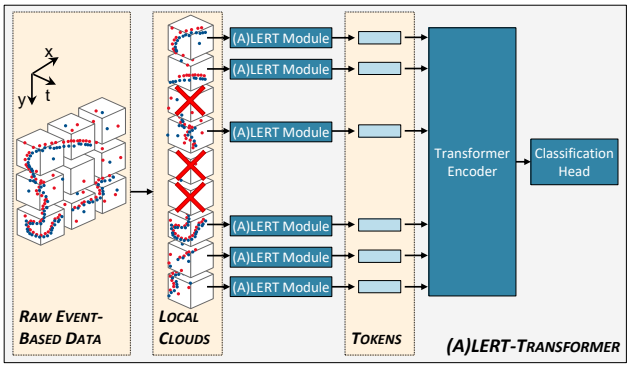

Figure 2. Overview of the (A)LERT-Transformer model. The event stream is spatially divided into local event clouds. The red crosses indicate the non-active tokens based on their number of events. (A)LERT module converts them to individual high dimension features, which are fed to a Transformer and classifier head.

2.1. LERT: Synchronous Events-To-Tokens Conversion

2.1.1. Raw Events to Normalized Local Patches

The embedding part – which we call **L**earnt **E**mbedding for **R**eal-**T**ime Processing of Event-Based Data (LERT) Module – is built upon the PointNet architecture (Qi et al., 2017a). LERT spatially divides the input stream in local event clouds (patches) and converts each of them into individual feature vectors (a.k.a. tokens). This module is depicted in Figure 4. To extract patches from an event sequence, a grid in the (x, y)-plane is constructed, effectively dividing the space into same-sized groups of pixels with predefined coordinates. Events triggered in the same pixel-group make a patch. Hence, a local event cloud is simply a set of events, which is a subset of the original input event stream.

Once the events are organized in patches, the LERT module applies two transformations. First, all patches which do not contain enough events are filtered out. The filter mechanism is a simple threshold on the number of events present inside the local point cloud. The threshold value is fixed and selected via a hyperparameter search. It could be trained. We denote the removed patches and events they contain as *non-active*. This method was proposed by (Sabater et al., 2022) and allows to exploit the sparsity of the input event-based data. The remaining events and patches are referred to as *active* events and active patches. The second step is a trivial normalization, where the spatial coordinates (x, y) of the active events are scaled down to the range $[-1, 1]$ with respect to a patch size (*not* the full image size). Once this preprocessing of the events is finished, the active event coordinates inside each patch are represented as follows:

$$(t, x, y, p) \mapsto t \in [0,\ T],\ x \in [-1,\ 1],\\ y \in [-1,\ 1],\ p \in \{-1,\ 1\}; \tag{1}$$

T denotes the duration of the input event stream sample (*not* the full file duration). LERT operates in a fully synchronous manner which makes it compatible with widespread machine learning tools and frameworks, simplifying training. It takes as input a finite list of events whose length (Ne) depends on the event accumulation mode. Ne is either always the same (we call it *Constant Count Input Mode* (**CCIM**)), or varying and depending on the number of events triggered during a fixed time window (denoted *Constant Time Input Mode* (**CTIM**)), in which case T is always the same between two samples. Figure 3 provides a visualization of event data and the two possible input modes.

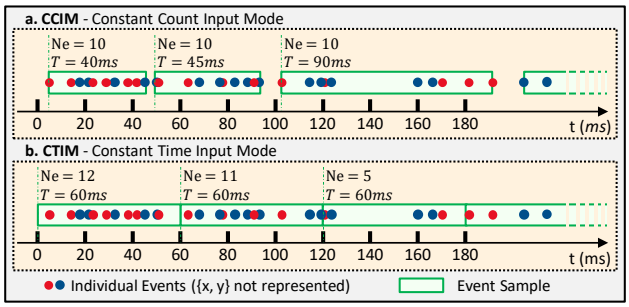

Figure 3. Schematic representation of the temporal sampling. In (a) Constant Count Input Mode (CCIM), the events are sampled from the continuous input event stream by splitting them into bins that include the same amount of events. In (b) Constant Time Input Mode (CTIM), each bin represents the same input duration. In both modes, every sample is then split into several subsets, based on the spatial coordinates of the events, as depicted in Figure 2. Note: blue and red points represent events with different polarities.

2.1.2. Normalized Local Patches to Tokens

The Feature Generator (FG) then converts each normalized input patch into an input token for the transformer model. The FG is shared between all the patches and consists in a customized shallow PointNet architecture. Every 4D active event contained in a single patch is processed through a shared MLP unit to generate a single high dimensional vector (of dimension c). The feature generator inherits the capacity of the PointNet to process an input of varying length. In our implementation, an MLP is composed of a succession of 1D-convolution layers.

Each 4D event is thus mapped to a high dimensional vector, and the Patch Feature is obtained by applying a channel-wise maximum operation on the resulting vectors. At this stage, all P active patches have been converted into P c-dimensional vectors, denoted as Patch Features (Fig. 5 (a)).

Normalizing and scaling the events with respect to the patch size results in information loss. In particular, the spatial origin of each local cloud in relation to the absolute (x,y) pixel grid is disregarded. Thus, the LERT module adds a positional embedding (*i.e.* a learnt linear embedding) to each Patch Feature, incorporating its spatial location within the predefined 2D grid. Positional embeddings supply the model with spatial distribution information about the tokens, they are learnt during training, and they are unique for every patch coordinate. A detailed schematic of the LERT embedding can be seen in Figure 4.

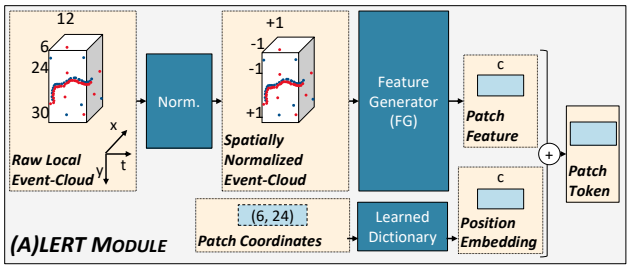

Figure 4. (A)LERT module: spatially local event cloud to token.

2.2. ALERT: Asynchronous Token Update

2.2.1. Introduction to the ALERT module

The LERT module is built for synchronous processing. As such, it does not take full advantage of the event-based nature of the data: it does not naturally update tokens when new events show up. Instead, it starts from scratch and recalculates all tokens for every new piece of data, which means it must collect events before it can process them.

We thus introduce the Asynchronous LERT (ALERT) Module, which is used during inference. This variation is designed to efficiently update the token sequence as new events occur (asynchronous processing), making it particularly suited for online inference of event-based data. To achieve this, the ALERT module requires three key modifications during training and/or inference: handling continuous values of time, updating tokens with each new event, and memorizing past information.

2.2.2. Handling Continuous Time Values

The LERT module alters the time coordinates of events in a sequence, shifting each input stream so that the first event starts at time 0. This approach introduces inefficiencies in real-time asynchronous processing, as it hinders the feasibility of recurrent or iterative processing of events. Our proposed solution consists in encoding the time value by using a periodic function. We achieve this by representing time with two sinusoidal waves: $t_x(t) = \alpha \cos(2\pi f t + \phi)$, and $t_y(t) = \alpha \sin(2\pi f t + \phi)$; taking inspiration from the positional encoding originally used in Transformers. The amplitude (α), frequency (f) and phase (ϕ) of these waves are constant and tuned for each application. The *Time Encoded* (**TE**) LERT (TELERT) Module applies this periodical wrap prior to the PointNet model. So, TELERT must be the module used during training if ALERT is used for inference. The only change made before the channel-wise maximum operation between the LERT and TELERT/ALERT modules is this time encoding (see Figure 5 b). Next, we present a method applied at token stage that emulates a sliding buffer in input during prediction.

2.2.3. Asynchronous Event-Based Token Update

The ALERT module is used for inference, and continuously updates the tokens as new events arrive, whenever k ($k \in \mathbb{Z}^+; k >= 1$) events have been triggered. When decomposing it, a token update requires both (1) adding information from new event(s) and (2) forgetting information from the oldest events.

Add Information: The k new event(s) are processed by TELERT. The resulting patch token is the channel-wise maximum between the previous patch token and the k new event feature vector(s).

Forgetting information using Old Maximum Value Decay (OMVD): The "age" of the last update from each tokens' channel is simply tracked by a counter. All channels whose associated counter channel is higher than a certain threshold undergo decay. All tokens are subject to this decay, not only the newly updated one, as the oldest information might be stored in different embeddings. The decay is applied in the form of an exponential ($\times e^{-\lambda}$), where the decay rate λ can be tuned for each model or scene characteristics, and could potentially be learned; the pseudocode can be found in Appendix A.1. This mechanism draws inspiration from the "leakage" concept in SNNs literature (Bouvier et al., 2019), but instead of applying it to all neurons/features of the model, we apply it only on tokens, to minimize compute.

2.2.4. Token Memorization and Activity Status

In ALERT, each new event triggers a corresponding patch token update. No raw event needs to be buffered. Instead, all tokens and associated age counter values are memorized throughout the entire runtime of the model, their patch being active or not. The non-activeness of the tokens is tracked using an event count for each patch. This counter is updated ($+1$ or -1) every time the token is updated (added

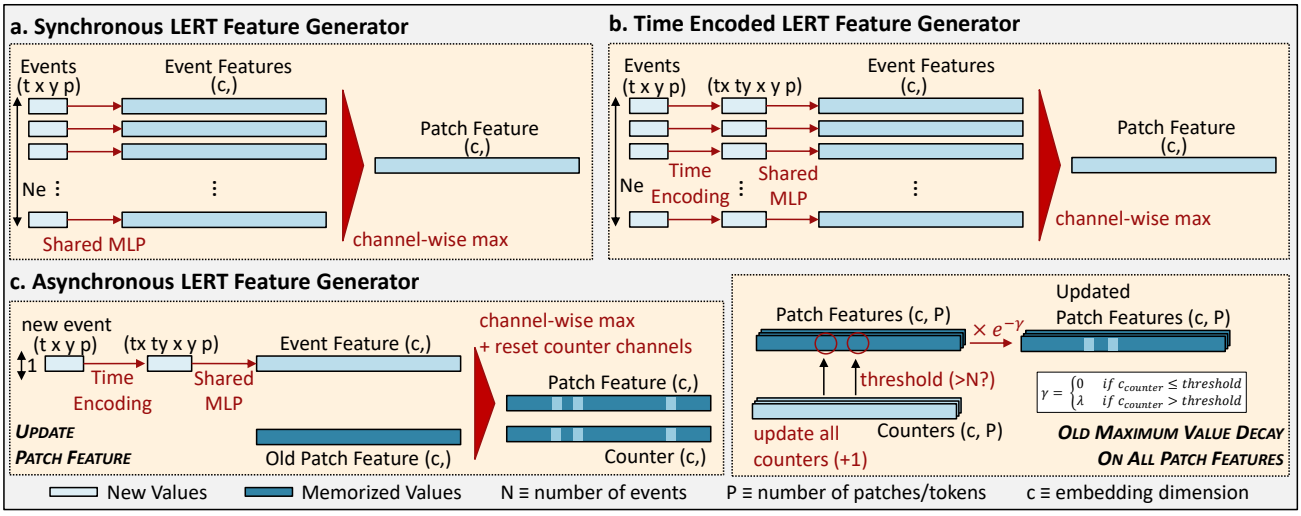

Figure 5. Overview of the modes of the (A)LERT feature generator (FG). The Time Encoded LERT FG (b) is used for training ALERT (c).

information or OMVD, respectively).

2.3. (A)LERT-Transformer

2.3.1. Transformer Encoder for Classification

To evaluate the proposed LERT and ALERT modules, we plug them to a Vision Transformer model trained from scratch for classification tasks. As our focus is to assess the functionality of the proposed learnt synchronous and asynchronous embedding modules, we deploy a stack of standard transformer encoders, as in the original ViT (Dosovitskiy et al., 2021). The final classification head simply consists of a single linear layer followed by a SoftMax activation function.

2.3.2. Synchronous Training and Asynchronous Inference

During training and for synchronous inference (using LERT or TELERT), all samples are processed independently from each other. So, classic GPU/TPU accelerated processing is done as usual. Once TELERT is trained, the embedding module can be converted to ALERT for inference by adding the event update functionalities (adding information and OMVD).

The ALERT version, used only for inference, can process events continuously. However, applying the feature generator on every single event without distributed processing is extremely time consuming. Hence, for simulation purposes, the asynchronous inference pipeline is assessed by updating and processing the tokens on-demand every Δt. This has no impact on the resulting accuracy, as the mathematics of the asynchronous update are ultimately not altered. Proof can be found in Appendix A.2.

3. Experiments

3.1. Experimental Setup

3.1.1. Datasets

We validate our approach through two classification tasks: action recognition and binary classification. For action recognition, we utilize the 11-class DVS128Gesture dataset (Amir et al., 2017), consisting of 2 to 6-seconds recordings from 29 users. For training (TE)LERT-Transformers, our input samples consist in Constant Count (CCIM) event streams of $Ne = 8192$ events each, with the events being randomly sampled in sequence from the original recordings. A file usually contains far more than 8192 events, thus ensuring a huge amount of possible random samples of Ne consecutive events. For binary classification, the Prophesee's N-Cars dataset (Sironi et al., 2018) is used. Each file is a 100ms recording containing either a car or a scene without a car ("background"). In this case our input training samples correspond to one recording, using Constant Time (CTIM) event streams with variable number of events. We use the original train and test splits for both datasets, and during asynchronous inference simulation with ALERT, we treat the entire test files as continuous sequences of events.

3.1.2. Performance Metrics

We describe here the metrics used to evaluate our hybrid asynchronous to synchronous network.

Accuracy. It is common practice in the literature to process several samples sequentially before settling on a decision (a class) (Sabater et al., 2022; Wang et al., 2019a) – which we denote *N-Voting Accuracy* (**NVA**) –, or to estimate the class over full files (Peng et al., 2023; Innocenti et al., 2021). Because our model does not rely on any recurrence between two consecutive input samples, it can process each sample

Table 1. Classification performance and complexity on the DVS128Gesture dataset. Event representation meanings: (F=Frames, T=Tokens, E=Events). *Online* refers to the capacity of achieving the said accuracy in a continuous inference paradigm. *Async.* refers to the event-driven processing nature of the network, where each event can be processed individually. TtA means *Time to Accuracy*, and it is the combination of *input accumulation time* ($\mathbf{t_{in}}$) and *inference time* ($\mathbf{t_p}$).

Model	Event Repr.	Async.	Online	TtA	Accuracy
3D-CNN (Innocenti et al., 2021)	F	✗	✗	File	99.6% (FVA)
GET (Peng et al., 2023)	F $\rightarrow$ T	✗	✗	File	97.9% (FVA)
PointNet++ (Wang et al., 2019a)	E	✗	✓	143ms	95.3% (NVA)
EventTransAct (Blegiers et al., 2023)	F $\rightarrow$ T	✗	✓	400ms+t_p	97.9% (NVA)
Event Tr. (Sabater et al., 2022)	F $\rightarrow$ T	✗	✓	480ms+t_p	94.4% (NVA)[2]
				24ms+t_p	79.8% (SA)[3]
CNN + LSTM (Innocenti et al., 2021)	F	✗	✓	500ms+t_p	97.7% (SA)
LERT-Tr. — RM	E $\rightarrow$ T	✗	✗	File	96.2% (FVA)
	E $\rightarrow$ T	✗	✓	132ms+t_p	88.6% (SA)
ALERT-Tr. — RM	E $\rightarrow$ T	✓	✗	File	94.1% (FVA)
	E $\rightarrow$ T	✓	✓	9.6ms (t_p)	84.6% (SA)
LERT-Tr. — LMM	E $\rightarrow$ T	✗	✗	File	92.0% (FVA)
	E $\rightarrow$ T	✗	✓	132ms+t_p	83.1% (SA)
ALERT-Tr. — LMM	E $\rightarrow$ T	✓	✗	File	89.2% (FVA)
	E $\rightarrow$ T	✓	✓	6.0ms (t_p)	72.9% (SA)

individually and independently. So, to compare with these works, we evaluate our model by voting over all predictions for each file, denoted *File Voting Accuracy* (**FVA**). However, to achieve ultra-low latency, providing a decision with every single sample is crucial. Hence, we also show the average accuracy over all samples analyzed independently, which we denote as *Sample Accuracy* (**SA**). Using SA results in lower accuracy, but enables ultra-low latency.

Time to Accuracy (TtA). The TtA represents the minimum amount of time needed to classify. This value depends on both the accuracy measurement method and the *total latency* ($\mathbf{t_{lat}}$) of the network (which equals *input accumulation time* ($\mathbf{t_{in}}$) plus *inference time* ($\mathbf{t_p}$)). In FVA, one needs to get predictions over the full file to reach a conclusion. So, the TtA is the total duration of a file. In an online setup (for real-world operation), the accuracy can be evaluated with different strategies, for instance using a sliding buffer for voting over N previous predictions (Wang et al., 2019a) (NVA). We target flexible and ultra-low latency and therefore consider the prediction of individual samples. So, we use the SA method where TtA equals t_{lat}.

Complexity. A good proxy for the time complexity is the number of operations per sample (FLOPs), which is indenpent of the employed hardware. We also report t_p for our model on an NVIDIA RTX 3080. The space complexity is associated to the number of parameters of a model.

3.1.3. Implementation Details

We present two models: a high-performing *Reference Model* (**RM**) and a *Low-Memory Model* (**LMM**) which is a reduced version of the same model. The RM consists in a 5-layer LERT module and a 4-layer Transformer Encoder with 8 attention heads and a token width of 512. The LMM features a small 2-layer LERT module and a (2 layers, 4 heads, 128-token width) Transformer Encoder. All models are implemented using `PyTorch` version 2.1. (Paszke et al., 2019). We train all networks with the cross-entropy loss and the LAMB optimizer (You et al., 2020).

3.2. Experimental Results

3.2.1. LERT-Transformer: Synchronous

Our RM model showcases noteworthy performance, with an FVA of 96.2% on the DVSGesture dataset (see Table 1). When measured with SA, the accuracy drops to 88.6%, but with a much lower latency of 141.6ms (t_{in}=132ms + t_p=9.6ms) on average. The RM is relatively complex, running at 1.299MFLOPs per event, and a total of 13.96M parameters for the full (A)LERT-Transformer model (see Table 2). The LERT module individually uses

1.218MFLOPs/event and 1.41M parameters. The LMM reduces this complexity by 24x to an impressive 0.566M parameters, with the LERT module requiring 264x less operation per event (4.0kFLOPs per event), and 0.04M parameters. The module together with the transformer runs at 7.4kFLOPs per event. LMM still shows a competitive FVA of 92.0% on the action recognition task. On the binary classification task with the N-Cars dataset, LMM achieves an accuracy of 85.6% with 0.54M parameters and an average of only 34.28M total FLOPs per $100ms$. Note that the average FLOP count per sample varies across different datasets, as the number of events per sample differs depending on sensor resolution and scene dynamics. On the contrary, the FLOPs for processing a single event with our model always remain the same, independently of the dataset.

Table 2. Model complexity on the DVS128Gesture dataset. Here, LERT (RM/LMM) refers only to the LERT module part of the associated model, while RM and LMM refer to the entire model (LERT-Transformer). t_p refers to the inference time, *i.e.* the time ALERT-Transformer takes to process a sample of 8192 events.

Model	#Param	FLOPs per		t_p
		event	sample	
LERT (RM)	1.41M	1.218M	8.83G	5.8ms
RM	13.96M	1.299M	9.42G	9.6ms
LERT (LMM)	0.04M	0.0040M	0.03G	3.9ms
LMM	0.57M	0.0074M	0.06G	6.0ms

3.2.2. ALERT-Transformer: Asynchronous

Moving from the synchronous LERT to the TELERT module, we noticed a negligible relative accuracy drop in FVA of 0.4% and 0% for the RM (95.8%) and LMM (92.0%) respectively (not shown in Table 1). This illustrates that time wrapping with sinusoidal encoding is an easy but efficient solution for bounded representation of time values. From synchronous LERT to ALERT inferences (adding TE and decay) the FVA accuracy relatively decreases by 2% and 3% for the RM (94.1%) and LMM (89.2%) versions, respectively. Nevertheless, this change in the model now allows for **asynchronous updating of the tokens**. The latency is entirely customizable as the tokens are continuously updated and can be processed on demand (the asynchronous to synchronous concept illustrated in Figure 1). The inference times t_p for processing $Ne = 8192$ events are 9.6ms and 6.0ms for the full RM and LMM, respectively. In the RM (LMM), the (A)LERT module takes 5.8ms (3.9ms) against the 3.8ms (2.1ms) for Transformer and Head. Simulating this asynchronous low-latency scenario, the RM and LMM models achieve SA accuracies of 84.6% and 72.9%, respectively. Because the (A)LERT module can be run event by event, we argue that using an event-driven asynchronous sparse AI accelerator could reduce the (A)LERT inference time to a few microseconds, thus enabling an on-demand ultra-low latency of 3.8ms (2.1ms) for the RM (LMM).

Table 3. Classification performance and latency on the N-Cars dataset. Comparison with state-of-the-art asynchronous models.

Model	Acc.	MFLOPs/ev
HOTS (Lagorce et al., 2017)	62.4%	14.0
HATS (Sironi et al., 2018)	90.2%	0.03
Async. Sparse CNNs (Messikommer et al., 2020)	**94.4%**	21.5
YOLE (Cannici et al., 2019)	92.7%	328.2
EST (Gehrig et al., 2019)	92.5%	1050
ALERT-Tr. — LMM (Ours)	85.6%	**0.0074**

3.2.3. Comparison with State-of-the-Art

Table 1 presents a comparative analysis with state-of-the-art for action recognition models. Many competitors lack the ability to process events asynchronously and cannot operate continuously (Online). PointNet++ (Wang et al., 2019a) still exhibits a low latency of 143ms for an NVA accuracy of 95.3% thanks to their sliding buffer voting and averaging strategy for improving accuracy. Note that our ALERT module could be deployed in their model, which would enable asynchronous PointNet++ processing instead of the current update every 25ms. EventNet (Sekikawa et al., 2019), a comparable model with asynchronous to synchronous capability, has limited performance on complex tasks as it applies a single PointNet for the full input event stream spatial dimension. Accuracy measures on gesture recognition has thus not been reported by the authors. This was one of our motivations towards splitting the input window along the spatial dimension. EventTransAct (Blegiers et al., 2023) processes individual clips containing 16 frames of 5ms. Every 5 clips, a prediction is obtained through mean aggregation, leading to a latency of 400ms, and a NVA of 97.9%. (Sabater et al., 2022) propose a recurrent Transformer model that integrates events into frames and processes them every 24ms. However, due to reccurence, predictions are originally obtained every 20 samples ($t_{in} = 480$ms), introducing significant latency. When using their model by requesting a prediction every 24ms (keeping the 20-sample recurrency) and applying SA the accuracy drops to 79.8%[3]. In this situation, our ALERT-Transformer RM outperforms theirs with an accuracy of 84.6%, but at the price of 1.299MFLOPs/event.

[2]For a fair comparison, we report the accuracy when evaluating their model on full files of DVSGesture test set. It leads to a 1.8% difference when compared with the original accuracy presented in (Sabater et al., 2022), where only the central 480ms of every file were used.

[3]The reported SA is obtained when the model makes a prediciton every 24ms, and latent vectors are reset every 20 samples.

Even though our LMM requires only 0.0074MFLOPs/event, in this case, it struggles with a 72.9% SA. But the different hyperparameter searches realized during our study revealed a major cost/accuracy tradeoff (see Appendix B.1).

Furthermore, when compared to state-of-the-art asynchronous models on the N-Cars binary classification task, our LMM showcases the lowest 0.0074MFLOPs/event complexity while achieving a competitive accuracy (see Table 3). This represents a 75% decrease in complexity compared to the previous state-of-the-art (Sironi et al., 2018), with only a 5.11% decrease in accuracy.

Overall, the proposed ALERT is a compelling event embedding module for Transformer models. It inherits event-based data properties, allows for highly flexible latency on-demand compute, and enables friction-less end-to-end learning.

3.3. Ablation Studies

We present the effects several hyperparameters have on the RM accuracy and complexity. Accuracies are depicted on the DVS Gesture dataset. We refer the reader to Appendix B.2 for a better visualization of the impact these parameters have on the patch sequences and the model.

Patch Size. The ALERT module spatially splits the input event stream in several smaller point clouds. Figure 6 depicts the variation of accuracy and total FLOPs for RM with respect to the patch size (height and width, in pixels). Smaller patches yield better models, but they are more complex and, as such, slower. The low accuracy of models with bigger patch size is predictable, as attention networks are known to perform worse with too short sequences. As the patch size increases, the number of available sub clouds – and consequential tokens – decreases, causing the poor performance of the Transformer classification network. Therefore, a tradeoff is necessary to find the optimum for each application and sensor resolutions.

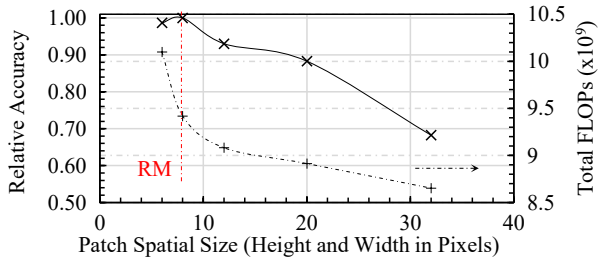

Figure 6. Influence of the patch size on accuracy (×) and complexity (+) (GFLOPs). Input resolution is 128 × 128 pixels. The RM is set as the baseline, hence has a 1.0 relative accuracy.

Active Patch Threshold. Understanding the role of the activation threshold in the (A)LERT-Transformer is key for using it efficiently. This threshold – defined in number of events per patch – determines the portion of patches considered as active. Interestingly, according to Figure 7 comparable accuracies are achieved with both low and high thresholds. Nevertheless, higher thresholds result in less complex models because the FLOPs per processed event remains constant, while less events are processed. In RM, the Transformer model is so small that the embedding part represents 94% of the total FLOPs of the model for $Ne = 8192$ events. Therefore, choosing a high threshold may be an important decision when a low complexity model with high accuracy is required.

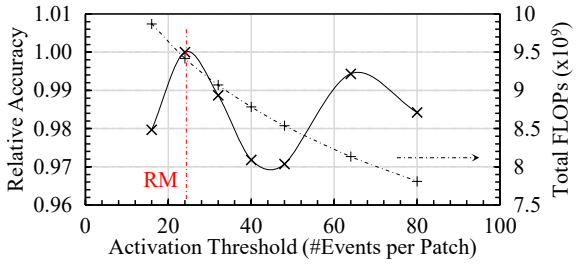

Figure 7. Influence of the patch activation threshold size on accuracy and complexity (GFLOPs). The RM is set as the baseline, hence has a 1.0 relative accuracy.

Input Sequence Length. The (TE)LERT module can be trained and evaluated in Constant Count or Constant Time input modes, and is thus suited for processing input sequences of varying lengths (Ne or Δ_t, respectively). In Figure 8a, we show how Ne impacts the CCIM model's performance. There seems to be an optimum point in between $Ne = 8$k and $Ne = 16$k events for the accuracy. However, the total complexity scales linearly with the number of processed events. Hence, our experiments were conducted using $Ne = 8192$, which yields 96.2% FVA. Similarly, for CTIM performance and complexity vary depending on Δ_t values (see Appendix B.4). The optimal point in this case is $\Delta_t = 180$ms, where the model reaches 88.5% SA and 96.5% FVA. The average time span of a DVS Gesture input sample in CCIM with $Ne = 8192$ fixed events, is of 130ms.

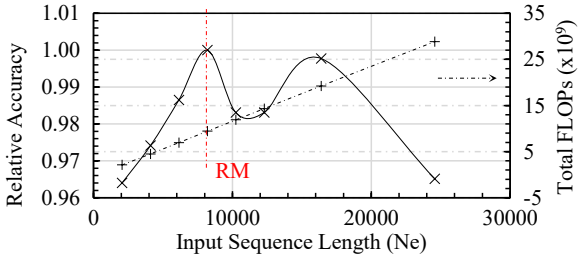

Figure 8. Influence of the input sequence length size on accuracy and complexity (GFLOPs). The RM is set as the baseline, hence has a 1.0 relative accuracy.

It is worth noting that using $\Delta_t = 130$ms in CTIM reaches 96.2% FVA, proving the equivalence between both input modes. Consequently, the input mode and its length should be adjusted based on task-specific requirements.

Old Maximum Value Decay. The decay rate is applied in the form of an exponential factor whenever a token channel has not been updated for a certain time. An optimal value exists for this decay rate, which should be tuned to simulate the behavior of the synchronous processing paradigm as truthfully as possible. Figure 9 shows the relationship between accuracy and decay rate for RM. Below the optimal rate, our model forgets past information too quickly; above the optimal value too much past information is kept, bringing confusion to the prediction. In the limit where information is not forgotten, *i.e.* $\lambda = 0$, the model reaches a sample accuracy of 25.9%, proving OMVD is a key component of the ALERT-Transformer.

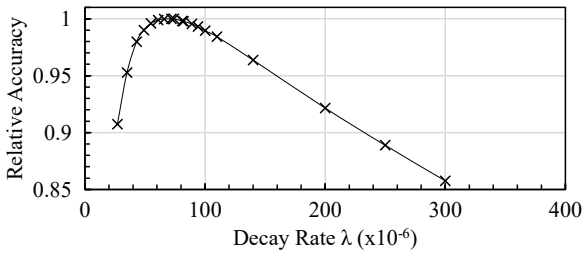

Figure 9. Decay rate influence on the ALERT accuracy during inference.

Positional embeddings. The encodings of the patch coordinates, provide information about the patch tokens spatial distribution in space. Training the LERT-Transformer Reference Model (RM) on the DVSGesture dataset without these positional encodings, leads to accuracies of 80.0% SA and 95.1% FVA (versus 88.6% SA, 96.2% FVA with positional embeddings). During inference they can be built as a simple Look-Up table, introducing a negligible overhead to the total cost of the model.

3.4. Limitations and Potential Applications

While the outcomes of this study showcase the viability of the proposed ALERT module for event data in Transformer-based computer vision pipelines, there are some limitations and room for improvement. The current validation focuses on a Transformer architecture for classification tasks. However, the tokens do not have to be used only by Transformers, and our method could be easily integrated with Convolutional Neural Networks (CNNs) or other architectures. Similarly, we are also working on the applicability of this learnt representation for other tasks than those used for benchmarking, such as generative ones. We see several directions for exploiting our research: trying to improve accuracy by adopting advanced strategies akin to concurrent processing of multiple samples (Peng et al., 2023) or using of memory tokens (Sabater et al., 2022); or exploring PointNet-only architectures based on the ALERT module (Wang et al., 2019a; Sekikawa et al., 2019).

The demonstrated learnt token sequence in the ALERT-Transformer model holds promise for diverse applications relying on computer vision, especially for edge solutions. Its adaptability across various pipelines, beyond transformer architectures, suggests utility in multiple tasks where event vision sensors offer advantages over conventional cameras. The flexibility of the on-demand data processing, and potential resulting energy-efficient capabilities of the ALERT-Transformer create opportunities for real-world deployment in scenarios using advanced systems for computer vision (always-on sensing to wake-up Application Processing Unit). Future research may explore integrating this hybrid asynchronous to synchronous representation into a broader array of vision tasks, leveraging its strengths for diverse real-time applications, and paving the way for potential hardware implementations. Also, it is worth noting that our pipeline could be integrated in multi-modal Transformers, seamlessly enabling mixed sensor computer vision.

4. Conclusions

Our research focused on addressing the fundamental challenge of efficiently processing sparse and asynchronous event-based data while leveraging its properties. Several, simple yet essential, contributions have been introduced: (1) end-to-end trained event-data to feature vectors conversion with the LERT module, and the modified Time Encoded LERT able to deal with continuous time values with negligible decrease of accuracy. (2) Seamless conversion to real-time event-driven processing with Asynchronous LERT, significantly reducing latency and enhancing the model's applicability to real-time applications. (3) An asynchronous sensing to on-demand synchronous processing framework: the ALERT-Transformer. An end-to-end system that ensures continuous and energy-efficient data processing for event-based vision sensors.

The ALERT-Transformer on the gesture recognition task achieves high accuracy (84.6%) with lowest ever latency (less than 9.6ms) during inference, outperforming comparative models with higher accuracy but slower processing. The reduced size model performs asynchronous binary classification at the lowest ever cost of 7.4kFLOPs/ev (to the best of our knowledge).

These contributions allow for significant advancement towards exploiting the potential of sparse heterogeneous multidimensional data, and lay the ground for further advancements in mixed sensors computer vision.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work. We shortly mention two of them here specifically related to using an Event-based technology. Our work aims at bridging the gap between asynchronous, event-based signal processing and conventional signal processing. On one hand, in doing so, it facilitates the use of such low-latency system, for example for autonomous systems, and make them more robust and possibly safe (e.g. for obstacle detection). On the other hand, accelerating the use of event-based sensing and processing in computer vision might lead to systems requiring less power, which, at scale, is relevant for the environment. We do not foresee any negative impact of this work.

References

Amir, A., Taba, B., Berg, D., Melano, T., McKinstry, J., Di Nolfo, C., Nayak, T., Andreopoulos, A., Garreau, G., Mendoza, M., Kusnitz, J., Debole, M., Esser, S., Delbruck, T., Flickner, M., and Modha, D. A low power, fully event-based gesture recognition system. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 7388–7397, 2017.

Blegiers, T., Dave, I., Yousaf, A., and Shah, M. Event-transact: A video transformer-based framework for event-camera based action recognition. 2023. doi: 10.1109/IROS55552.2023.10341740.

Bonazzi, P., Rüegg, T., Bian, S., Li, Y., and Magno, M. Tiny-Tracker: Ultra-Fast and Ultra-Low-Power Edge Vision In-Sensor for Gaze Estimation. In *2023 IEEE SENSORS*, pp. 1–4, October 2023. doi: 10.1109/SENSORS56945.2023.10325167. URL https://ieeexplore.ieee.org/abstract/document/10325167. ISSN: 2168-9229.

Bouvier, M., Valentian, A., Mesquida, T., Rummens, F., Reyboz, M., Vianello, E., and Beigne, E. Spiking Neural Networks Hardware Implementations and Challenges: A Survey. *ACM Journal on Emerging Technologies in Computing Systems*, 15(2):22:1–22:35, April 2019. ISSN 1550-4832. doi: 10.1145/3304103. URL https://doi.org/10.1145/3304103.

Bouvier, M., Valentian, A., and Sicard, G. Scalable Pitch-Constrained Neural Processing Unit for 3D Integration with Event-Based Imagers. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pp. 385–390, December 2021. doi: 10.1109/DAC18074.2021.9586099. URL https://ieeexplore.ieee.org/abstract/document/9586099. ISSN: 0738-100X.

Caccavella, C., Paredes-Vallés, F., Cannici, M., and Khacef, L. Low-power event-based face detection with asynchronous neuromorphic hardware, December 2023. URL http://arxiv.org/abs/2312.14261. arXiv:2312.14261 [cs].

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

Chen, J., Shi, H., Ye, Y., Yang, K., Sun, L., and Wang, K. Efficient human pose estimation via 3d event point cloud. In *Proceedings of the 2022 International Conference on 3D Vision (3DV)*, pp. 1–10, 2022.

Dampfhoffer, M., Mesquida, T., Valentian, A., and Anghel, L. Are snns really more energy-efficient than anns? an in-depth hardware-aware study. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 7(3):731–741, 2023. doi: 10.1109/TETCI.2022.3214509.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. An image is worth 16x16 words: Transformers for image recognition at scale. In *Proceedings of the 9th International Conference on Learning Representations (ICLR 2021)*, 2021.

Finateu, T., Niwa, A., Matolin, D., Tsuchimoto, K., Mascheroni, A., Reynaud, E., Mostafalu, P., Brady, F., Chotard, L., LeGoff, F., Takahashi, H., Wakabayashi, H., Oike, Y., and Posch, C. 5.10 A 1280×720 Back-Illuminated Stacked Temporal Contrast Event-Based Vision Sensor with 4.86μm Pixels, 1.066GEPS Readout, Programmable Event-Rate Controller and Compressive Data-Formatting Pipeline. In *2020 IEEE International Solid- State Circuits Conference - (ISSCC)*, pp. 112–114, San Francisco, CA, USA, February 2020. IEEE. ISBN 978-1-72813-205-1. doi: 10.1109/ISSCC19947.2020.9063149. URL https://ieeexplore.ieee.org/document/9063149/.

Gallego, G., Delbrück, T., Orchard, G., Bartolozzi, C., Taba, B., Censi, A., Leutenegger, S., Davison, A. J., Conradt, J., Daniilidis, K., et al. Event-based vision: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 44(1):154–180, 2020.

Garrett, D., Park, Y. S., Kim, S., Sharma, J., Huang, W., Shaghaghi, M., Parthasarathy, V., Gibellini, S., Bailey, S., Moturi, M., Vorenkamp, P., Busch, K., Holleman, J., Javid, B., Yousefi, A., Judy, M., and Gupta,

A. A 1mW Always-on Computer Vision Deep Learning Neural Decision Processor. In *2023 IEEE International Solid-State Circuits Conference (ISSCC)*, pp. 8–10, February 2023. doi: 10.1109/ISSCC42615.2023.10067588. URL https://ieeexplore.ieee.org/document/10067588. ISSN: 2376-8606.

Gehrig, D., Loquercio, A., Derpanis, K. G., and Scaramuzza, D. End-to-end learning of representations for asynchronous event-based data. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

Innocenti, S., Becattini, F., Pernici, F., and Del Bimbo, A. Temporal binary representation for event-based action recognition. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pp. 10426–10432, Los Alamitos, CA, USA, jan 2021. IEEE Computer Society. doi: 10.1109/ICPR48806.2021.9412991.

Kaufmann, E., Bauersfeld, L., Loquercio, A., Müller, M., Koltun, V., and Scaramuzza, D. Champion-level drone racing using deep reinforcement learning. *Nature*, 620(7976):982–987, August 2023. ISSN 1476-4687. doi: 10.1038/s41586-023-06419-4. URL https://www.nature.com/articles/s41586-023-06419-4. Number: 7976 Publisher: Nature Publishing Group.

Lagorce, X., Orchard, G., Galluppi, F., Shi, B. E., and Benosman, R. B. Hots: A hierarchy of event-based time-surfaces for pattern recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(7):1346–1359, 2017. doi: 10.1109/TPAMI.2016.2574707.

Maqueda, A. I., Loquercio, A., Gallego, G., Garcia, N., and Scaramuzza, D. Event-based vision meets deep learning on steering prediction for self-driving cars. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. IEEE, jun 2018. doi: 10.1109/cvpr.2018.00568.

Messikommer, N., Gehrig, D., Loquercio, A., and Scaramuzza, D. Event-based asynchronous sparse convolutional networks. In Vedaldi, A., Bischof, H., Brox, T., and Frahm, J. (eds.), *Procedding of the 16th European Conference on Computer Vision (ECCV 2020)*, volume 12353 of *Lecture Notes in Computer Science*, pp. 415–431, Glasgow, UK, 2020. Springer.

Moosmann, J., Bonazzi, P., Li, Y., Bian, S., Mayer, P., Benini, L., and Magno, M. Ultra-Efficient On-Device Object Detection on AI-Integrated Smart Glasses with TinyissimoYOLO, November 2023. URL http://arxiv.org/abs/2311.01057. arXiv:2311.01057 [cs].

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf.

Peng, Y., Zhang, Y., Xiong, Z., Sun, X., and Wu, F. Get: Group event transformer for event-based vision. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 6038–6048, October 2023.

Qi, C. R., Su, H., Mo, K., and Guibas, L. J. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 77–85, 2017a.

Qi, C. R., Yi, L., Su, H., and Guibas, L. J. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *Advances in neural information processing systems*, 30, 2017b.

Sabater, A., Montesano, L., and Murillo, A. C. Event transformer. a sparse-aware solution for efficient event data processing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW 2022)*, pp. 2677–2686, June 2022.

Sabater, A., Montesano, L., and Murillo, A. C. Event transformer$^+$. a multi-purpose solution for efficient event data processing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(12):16013–16020, 2023.

Sekikawa, Y., Hara, K., and Saito, H. Eventnet: Asynchronous recursive event processing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

Sironi, A., Brambilla, M., Bourdis, N., Lagorce, X., and Benosman, R. Hats: Histograms of averaged time surfaces for robust event-based object classification. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1731–1740, 2018.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need. In Guyon, I., von Luxburg, U., Bengio, S., Wallach, H. M., Fergus, R., Vishwanathan, S. V. N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 30 (NIPS 2017)*, pp. 5998–6008, Long Beach, CA, USA, 2017. Curran Associates, Inc.

Wang, Q., Zhang, Y., Yuan, J., and Lu, Y. Space-time event clouds for gesture recognition: From rgb cameras to event cameras. In *2019 IEEE Winter Conference on Applications of Computer Vision (WACV 2019)*, pp. 1826–1835, 2019a.

Wang, Y., Sun, Y., Liu, Z., Sarma, S. E., Bronstein, M. M., and Solomon, J. M. Dynamic graph cnn for learning on point clouds. *ACM Transactions on Graphics (TOG)*, 38 (5), oct 2019b. ISSN 0730-0301.

Wang, Z., Hu, Y., and Liu, S.-C. Exploiting spatial sparsity for event cameras with visual transformers. In *2022 IEEE International Conference on Image Processing (ICIP)*, pp. 411–415, 2022.

You, Y., Li, J., Reddi, S., Hseu, J., Kumar, S., Bhojanapalli, S., Song, X., Demmel, J., Keutzer, K., and Hsieh, C.-J. Large Batch Optimization for Deep Learning: Training BERT in 76 minutes, January 2020. URL http://arxiv.org/abs/1904.00962. arXiv:1904.00962 [cs, stat].

Yu, X., Tang, L., Rao, Y., Huang, T., Zhou, J., and Lu, J. Point-BERT: Pre-training 3D Point Cloud Transformers with Masked Point Modeling. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 19291–19300, June 2022. doi: 10.1109/CVPR52688.2022.01871. URL https://ieeexplore.ieee.org/document/9880161. ISSN: 2575-7075.

Zhao, H., Jiang, L., Jia, J., Torr, P. H., and Koltun, V. Point transformer. In *Proceedings of the 2021 International Conference on Computer Vision (ICCV 2021)*, pp. 16259–16268, 2021.

Zhu, A. Z., Yuan, L., Chaney, K., and Daniilidis, K. Ev-flownet: Self-supervised optical flow estimation for event-based cameras. In Kress-Gazit, H., Srinivasa, S. S., Howard, T., and Atanasov, N. (eds.), *Robotics: Science and Systems XIV, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA, June 26-30, 2018*, 2018.

A. ALERT Module: Proofs and Definitions

A.1. Algorithm: Asynchronous Event-Based Token Updates

The following algorithm presents the mathematical operations performed during a single token update with the ALERT module in inference mode. This update corresponds to a token, G_a, from a patch that has a new event, e_{new}, whose information needs to be incorporated. The Old Maximum Value Decay strategy is also applied to this token. OMVD consists in applying a decay to old values and, although not shown in the algorithm, it is applied to in all tokens, not only G_a. Below, the function $f(e)$ represents the shared MLP that builds a feature for each event; N is the threshold for OMVD to be applied; and $counter_a$ keeps track of the latest updates in all channels of G_a.

Algorithm 1 Asynchronous Event-Based Update of a Patch Token with Old Maximum Value Decay

Require: e_{new}, $counter_a$, G_a, N
  for C in embbedings dimension **do**
    $G_a[C] \leftarrow \max(G_a[C],\ f(e_{new})[C])$
    if $G_a[C]\ ==\ f(e_{new})[C]$ **then**
      $counter_a[C] \leftarrow 0$
    else
      $counter_a[C] \leftarrow counter_a[C] + 1$
      if $counter_a[C] > N$ **then**
        $G_a[C] \leftarrow G_a[C] \times \exp(-\lambda)$
      end if
    end if
  end for

A.2. Proof: Iterative Updating of Tokens

Global features in the proposed model are obtained from the channel-wise maximum values amongst all the event features in a cloud. Let us demonstrate the equivalence between updating the token channels as each events arrives, and creating the tokens by applying the max function to several available new events at once (TELERT module).

Consider a sequence of values $x_1, x_2, \ldots, x_N$, and let M represent the maximum value in the sequence. In the LERT token creation paradigm, the maximum value is obtained by applying the max function to all values at once:

$$M_{\text{all}} = \max(x_1, x_2, \ldots, x_N) \tag{2}$$

On the other hand, with the ALERT token creation paradigm, the maximum value is obtained by applying the max function iteratively as new values arrive. Define M_k as the maximum value after considering the first k values:

$$M_k = \max(x_1, x_2, \ldots, x_k) \tag{3}$$

After adding a new value x_{k+1} to the sequence, the updated maximum is:

$$M_{k+1} = \max(M_k, x_{k+1}) \tag{4}$$

Now, let us show that $M_{\text{all}} = M_N$, demonstrating the equivalence of the two approaches.

$$\begin{aligned} M_{\text{all}} &= \max(x_1, x_2, \ldots, x_N) \\ &= \max(\max(x_1, x_2, \ldots, x_{N-1}), x_N) \\ &= \max(\max((\max(\max(x_1, x_2), x_3), \ldots), x_{N-1}), x_N) \\ &= M_N \end{aligned}$$

We have now shown that applying the max function to all values at once is equivalent to applying it iteratively as new values arrive. Furthermore, we would like to proof that the Old Maximum Value Decay strategy leads to congruent results.

Consider a global feature where a specific channel, M, has not been updated for a long time, meaning that the channel is taking the information from an old event. In this case, with non-iterative processing, this channel's value would be updated, as the old event feature would not be included in the max operation anymore.

$$M = \max(x_0, x_1, ..., x_N) \quad \rightarrow x_{N+1} \text{ not used} \tag{5}$$

Here, x_i represents the corresponding channel from each event feature. Now, if we want to do iterative updating of the tokens, we cannot recompute the max over all values every time. Thus, we find a decay strategy where old channel values lose their relevance over time, instead of completely removing these values.

$$M = \max(x_0, x_1, ..., x_N, x_{N+1} \times e^{-\lambda}) \tag{6}$$

As time continues and new events arrive, the oldest events values lose their relevance exponentially.

$$X = \max(x_0, x_1, ..., x_N, x_{N+1} \times e^{-\lambda}, x_{N+2} \times e^{-2\lambda}, ..., x_{N+l} \times e^{-l\lambda}) \tag{7}$$

The decay term ensures that the influence of events decreases exponentially over time. Thus, in the limit where time approaches infinity, channels from old event features will tend to zero, not being able to "win" in the max function.

The ALERT module methodology comes from merging the iterative addition of information and this decay strategy. The combination of both approaches slightly alters the results from completely computing tokens from scratch, but the difference is negligible when compared to the gains the ALERT module provides.

$$\begin{aligned} X_t &= \max(x_{new}, X_{t-1}) && \text{if } counter_c < \text{ threshold} \\ X_t &= \max(x_{new}, X_{t-1} \times e^{-\lambda}) && \text{if } counter_c >= \text{ threshold} \end{aligned}$$

B. (A)LERT-Transformer: In-Depth Ablation Study

The following figures and tables show in a visual and detailed manner the influence several hyperparameters have on the ALERT-Transformer model.

B.1. Patch Size, Activation Rate and Sequence Length

Table 4. **ALERT Hyperparameter Analysis.** Influence of several key embedding hyperparameters on the performance and complexity of the LERT-Transformer model trained on the DVS Gesture Dataset for action recognition.

(a) Sequence Length

	Relative Accuracy	Total FLOPs
2048	0.963%	2.092G
4096	0.973%	4.492G
6144	0.985%	6.949G
8192	1.000%	9.419G
10240	0.982%	11.87G
12288	0.982%	14.32G
16384	0.997%	19.24G
24576	0.964%	28.86G

(b) Voxel's Spatial Size

	Relative Accuracy	Total FLOPs
$[6, 6]$	0.985%	10.10G
$[8, 8]$	1.000%	9.419G
$[12, 12]$	0.973%	9.082G
$[20, 20]$	0.985%	8.914G
$[32, 32]$	0.999%	8.653G

(c) Voxel's Activation Rate

	Relative Accuracy	Total FLOPs
0.50	0.979%	9.871G
0.75	1.000%	9.419G
1.00	0.988%	9.071G
1.25	0.971%	8.785G
1.50	0.970%	8.538G
2.00	0.993%	8.134G
2.50	0.983%	7.811G

B.2. Visualization of the Patch Size and Sequence Length

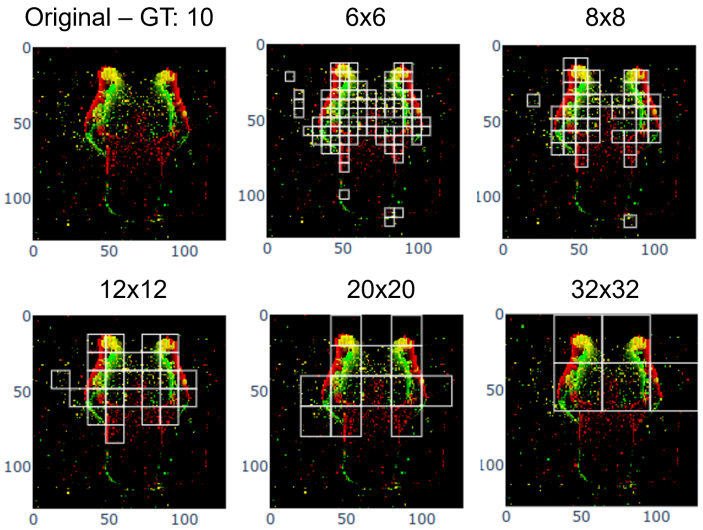

Figure 10. **Patch Size Visualization:** A simple 2D representation of the same sample with its corresponding active voxels for different patch sizes. This representation allows for a better understanding of the spatial division of the event stream. Each drawn patch corresponds to an active voxel, which events will be processed through the feature generator to obtain a token. The smaller the patch size, the less information a single voxel contains. As a result, with the same processing pipeline, a more detailed representation can be obtained when using smaller patches. The need exists to find a trade-off between the complexity of the model, and the level of detail of the token representations. The sample consists of a stream of 8192 events from the DVS128 Gesture Dataset.

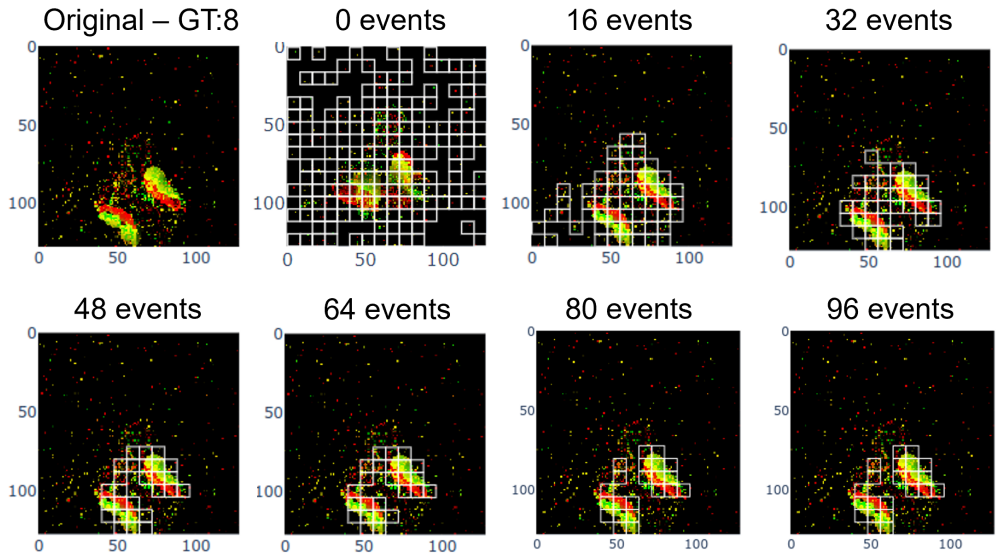

Figure 11. **Patch Activation Threshold Visualization:** A simple 2D representation of the same sample with its corresponding active patches for the same patch size, but different activation thresholds. This simple representation allows for a straightforward understanding of the necessity of discarding certain patches. Event-based data is inherently sparse, and this should be leveraged when processing it. Eventhough the model performs similarly with different thresholds, it is important to note that processing almost empty patches does not add any relevant information while adding a great computational cost. Discarding sub clouds where no movement occurs removes noise and allows the model to decrease its complexity. The sample consists of a stream of 8192 events from the DVS128 Gesture Dataset.

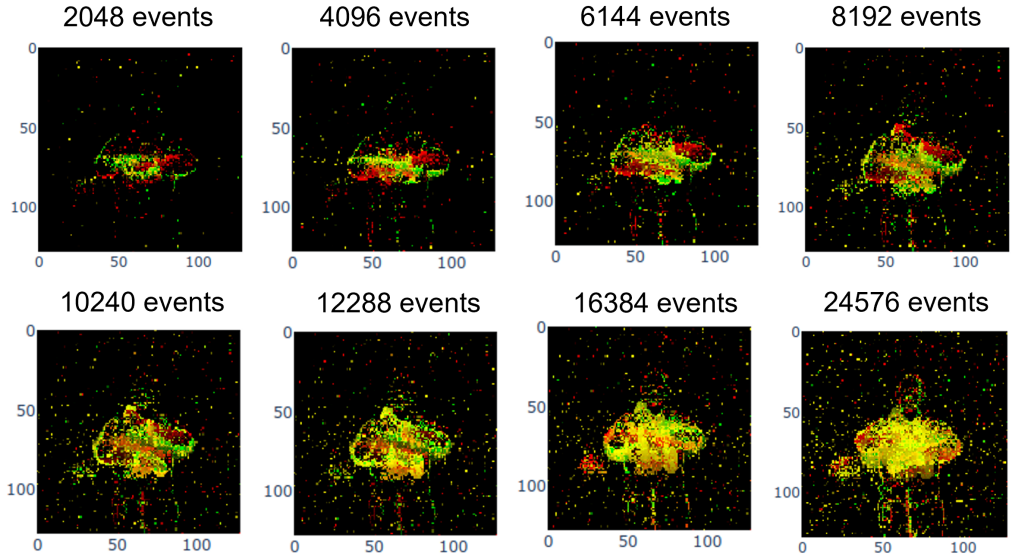

Figure 12. **Sequence Length Visualization:** A simple 2D representation of samples formed by different number of events. All of the samples share the first event but extend to different lengths. The samples consist of streams of events from the DVS128 Gesture Dataset. Although this frame representation of the samples is *not* used at any point in the proposed pipeline, it serves as visualization of the different sample lengths and the amount of information they carry. This analysis highlights the necessity of selecting input samples appropriately to the dataset and task requirements, as too little information may limit the performance, while too much may be too difficult to learn with a simple model.

B.3. Feature Generator Network Depth

The optimal number of layers for the Feature Generator's MLP strongly depends on other hyperparameters of the model. This value has a direct impact on the model's memory footprint, as it changes the number of parameters of the model. The Feature Generator has two important tunable hyperparameters: the number of layers of the MLP (depth) and the number of neurons in each of these layers. The input and output of the FG have fixed dimensions, being a 4-dimensional point and a 512-dimensional vector, respectively. However, the dimension and number of in-between hidden layers can vary depending on the model's requirements. Three different hyperparameters are used during our implementation to define this configuration:

- *Depth:* Number of layers of the MLP. In the implementation, each of these layers comprises a 1D-convolutional layer, batch normalization and an activation function (ReLU).
- *Base channels:* Number of neurons of the first hidden layer.
- *Expansion Factor (β):* Determines the increment of neurons from one hidden layer to the next as follows: $hidden_i = \beta \times hidden_{i-1}$.

Figure 13 provides an insight on how the modification of these values alters the performance and memory footprint of the model. In general, the fewer layers and number of neurons, the less accurate a model is. However, this decrease in accuracy is quite slow. On the contrary, reducing the number of hidden layers, and the neurons per layer, decreases the size of the model exponentially. The relationship between the metrics and the MLP's hyperparameters highlights the necessity of a trade-off between complexity and accuracy for each model, and the optimal solution will depend on the specific application requirements. Our RM and LMM provide two cases where this trade-off are considered, but further models are to be explored.

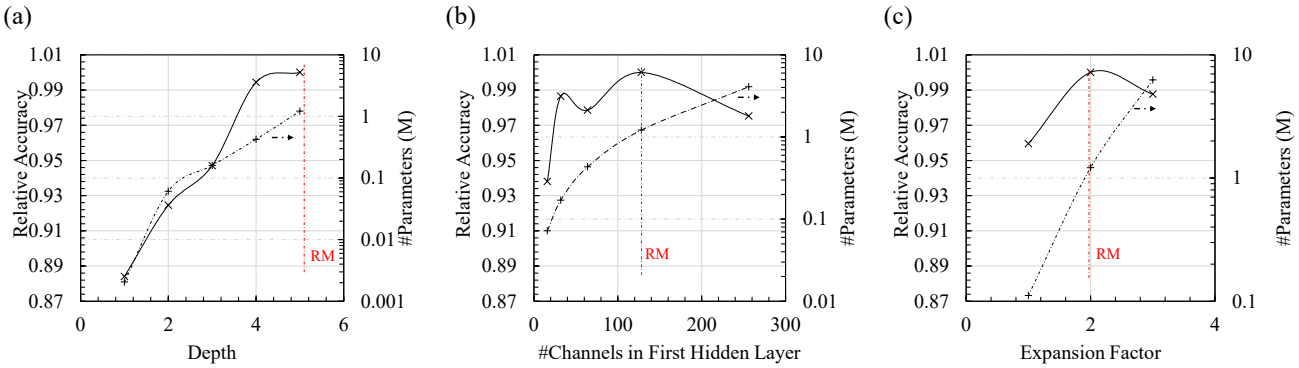

Figure 13. **Feature Generator's Depth impact on the Model's Performance.** Overall,the depth of the (A)LERT module has a major influence on the accuracy of the network. The deeper the most accurate. But at the same time, network complexity increases. So a tradeoff needs to be made regarding the depth. On the other hand, the projection dimension does not have the same relationship with accuracy. Indeed, it seems that one should avoid using a too small projection width, but an optimum can be found. The same seems to be true for the expansion factor. The two later points provide an optimism regarding the search for an optimal accuracy to cost ratio at a given depth. Note: the vertical red lines illustrate the RM configuration.

B.4. Constant Time Input Mode

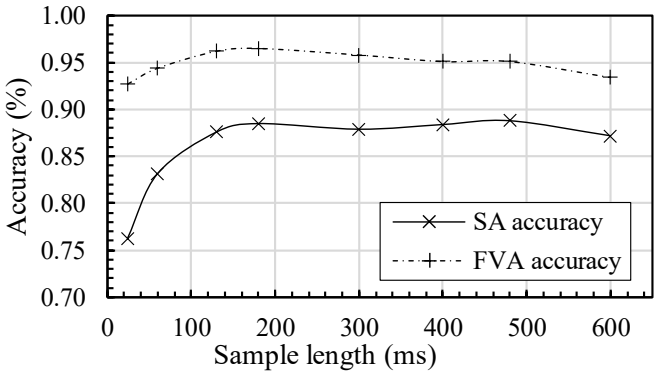

Figure 14. **Duration of an Input Sample (in CTIM) impact on the Model's Performance.**