Event-based optical flow on neuromorphic processor
ANN vs. SNN comparison based on activation sparsification

Xu, Yingfu; Tang, Guangzhi; Yousefzadeh, Amirreza; de Croon, Guido C.H.E.; Sifalakis, Manolis

DOI
10.1016/j.neunet.2025.107447

Publication date
2025

Document Version
Final published version

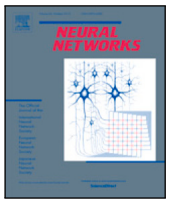
Published in
Neural Networks

Neural Networks 188 (2025) 107447

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet



Full Length Article

Event-based optical flow on neuromorphic processor: ANN *vs.* SNN comparison based on activation sparsification

Yingfu Xu [a,*,1], Guangzhi Tang [b], Amirreza Yousefzadeh [c], Guido C.H.E. de Croon [d], Manolis Sifalakis [a,2]

[a] Hardware-Efficient Artificial Intelligence Team, Stichting imec Nederland, High Tech Campus 31, Eindhoven, 5656 AA, The Netherlands
[b] Department of Advanced Computing Sciences, Maastricht University, Paul-Henri Spaaklaan 1, Maastricht, 6229 EN, The Netherlands
[c] Faculty of EEMCS, University of Twente, Drienerlolaan 5, Enschede, 7522 NB, The Netherlands
[d] Faculty of Aerospace Engineering, Delft University of Technology, Kluyverweg 1, Delft, 2629 HS, The Netherlands

ARTICLE INFO

Keywords:
Neuromorphic computing
Activation sparsification
Spiking neural networks
Benchmarking
Event-based optical flow

ABSTRACT

Spiking neural networks (SNNs) for event-based optical flow are claimed to be computationally more efficient than their artificial neural networks (ANNs) counterparts, but a fair comparison is missing in the literature. In this work, we propose an event-based optical flow solution based on activation sparsification and a neuromorphic processor, SENECA. SENECA has an event-driven processing mechanism that can exploit the sparsity in ANN activations and SNN spikes to accelerate the inference of both types of neural networks. The ANN and the SNN for comparison have similar low activation/spike density (~5%) thanks to our novel sparsification-aware training. In the hardware-in-loop experiments designed to deduce the average time and energy consumption, the SNN consumes 44.9ms and 927.0μJ, which are 62.5% and 75.2% of the ANN's consumption, respectively. We find that SNN's higher efficiency is attributed to its lower pixel-wise spike density (43.5% *vs.* 66.5%) that requires fewer memory access operations for neuron states.

1. Introduction

Estimating optical flow using an event camera is robust to motion blur and varying illumination thanks to the event stream that captures pixel brightness changes asynchronously in high dynamic ranges. Similar to most computer vision tasks, more and more accurate event-based optical flow estimation has been achieved by deep neural networks, referred to as artificial neural networks (ANNs) in this paper (Gallego, Gehrig, & Scaramuzza, 2019; Gehrig, Millhäusler, Gehrig, & Scaramuzza, 2021; Paredes-Vallés, Scheper, De Wagter and de Croon, 2023; Shiba, Aoki, & Gallego, 2022; Zhu & Yuan, 2018; Zhu, Yuan, Chaney, & Daniilidis, 2019). Spiking neural networks (SNNs) have different operating mechanisms from ANNs that are closer to the biological neural circuits. Instead of using scalar activations to communicate between neurons, neurons of an SNN maintain their membrane voltages (neuron states) across time steps and communicate by binary activations (spikes). An event camera is known as a neuromorphic sensor given its sparse, asynchronous, and spike-form measurements. Therefore, SNN potentially fits the event camera better. Researchers have been working on using SNNs for event-based optical flow (Chaney, Panagopoulou, Lee, Roy, & Daniilidis, 2021; Cuadrado, Rançon, Cottereau, Barranco, & Masquelier, 2023; Hagenaars, Paredes-Vallés, & De Croon, 2021; Kosta & Roy, 2023; Lee et al., 2020; Ponghiran, Liyanagedera, & Roy, 2023; Schnider et al., 2023). Although the gap is small, SNNs predict less accurate optical flow than ANNs. On the other hand, it is widely believed that SNNs have advantageous computational efficiency.

Because spikes are binary, SNNs require only accumulate (AC) operations while ANNs require multiply-and-accumulate (MAC) operations for a synaptic operation, *i.e.* the integration of an input activation into the neuron (Lemaire, Miramond, Bilavarn, Saoud, & Abderrahmane, 2022). In addition, SNN spikes have a sparse nature. Neuromorphic processors exploit the sparsity by skipping unnecessary synaptic operations caused by zero inputs to neurons (Davies et al., 2021; Furber & Bogdan, 2020; Modha et al., 2023; Tang, Vadivel et al., 2023). Based on the two facts above, works (especially algorithmic ones) compared the number of synaptic operations of ANN and SNN to support the claim that SNNs are computationally more efficient (Deng et al., 2020; Sengupta, Ye,

* Corresponding author.
E-mail addresses: yingfu.xu.94@gmail.com (Y. Xu), guangzhi.tang@maastrichtuniversity.nl (G. Tang), a.yousefzadeh@utwente.nl (A. Yousefzadeh), g.c.h.e.decroon@tudelft.nl (G.C.H.E. de Croon), manolis.sifalakis@innatera.com (M. Sifalakis).
[1] Now at Amazon.com Inc.
[2] Now at Innatera Nanosystems B.V.

https://doi.org/10.1016/j.neunet.2025.107447
Received 26 July 2024; Received in revised form 9 February 2025; Accepted 27 March 2025
Available online 7 April 2025

Wang, Liu, & Roy, 2019). Many works of SNNs for event-based optical flow took the measured data from Horowitz (2014) that MAC is 5.1× more expensive than AC to calculate energy cost (Chaney et al., 2021; Hagenaars et al., 2021; Kosta & Roy, 2023; Lee et al., 2020; Ponghiran et al., 2023).

However, only considering the difference between the computational costs of AC and MAC is not convincing. More comprehensive ANN *vs.* SNN comparison works keep emerging but there are three major ignored facts in existing comparisons, regarding the task, processor, and activation sparsity of ANNs. First, the tasks in ANN *vs.* SNN comparisons were often image classification, where converting the scalar pixel intensity into a series of spikes is required and thus SNNs need multiple forward-passes to produce a prediction while ANNs need only one (Lee, Kim, Lee, Baek, & Kim, 2021; Lemaire et al., 2022; Sengupta et al., 2019). Besides, the SNNs were often converted from a trained ANN. There has been no regression task discussed yet. Second, either only ANNs were benchmarked on an accelerator (Dampfhoffer, Mesquida, Valentian, & Anghel, 2022) or the processors were different for SNNs and ANNs (Lemaire et al., 2022), harming the fairness of comparison. Third, although zero activations of ANNs were considered in energy calculation in works (Dampfhoffer et al., 2022; Lee et al., 2021), there is no work yet comparing SNNs with sparsified ANNs, even if there are fruitful research results in increasing and exploiting ANN activation sparsity (Cao et al., 2019; Georgiadis, 2019; Grimaldi, Ganji, Lazarevich, & Sah, 2023; Kurtz et al., 2020; Li et al., 2023; Runwal, Pedapati, & Chen, 2023; Yang, Mao, Wang, & Hai, 2021).

In this work, we aim to address the above problems by proposing an efficient neuromorphic solution for event-based optical flow estimation, which is a representative regression task, filling in the gaps in existing works. It has wide application fields including power-sensitive mobile robots (Paredes-Vallés, Hagenaars et al., 2023) and always-on sensing devices, where energy-efficient neuromorphic processors are preferred. Unlike a frame-based camera filming pixel intensity, an event camera allows SNNs to directly use its spike-form measurements. Therefore, both ANNs and SNNs need only one forward-pass for a prediction. To address the issue of different processors for ANN and SNN, we adopt a neuromorphic processor, SENECA (Tang, Vadivel et al., 2023; Yousefzadeh et al., 2022). SENECA supports both network types with the same processing logic, enhancing the fairness of the comparison. As for the ANN's activation sparsity, we sparsify both ANN activations and SNN spikes before comparison. To do so, we propose an effective activation sparsification approach requiring only one training attempt. Experimental data of performance, *i.e.* network prediction accuracy and cost of time and energy, measured onboard the neuromorphic processor are analyzed to find out how activation/spike density and spatial distribution affect networks' performance. This is the first work showing an SNN's advantageous energy and time efficiency over an ANN in a regression task of event-based vision by a fair comparison supported by experimental measurements on a neuromorphic processor. In the rest of this paper, we use *ac./sp.* to refer to ANN activation and SNN spike when a statement applies to both of them, for simplicity. The term *ac./sp.* density indicates the percentage of non-zero elements in the output tensors of layers. The main contributions are summarized as follows:

- We propose using an activation function with a channel-level thresholding mechanism and surrogate gradience to train the thresholds. This approach significantly reduces *ac./sp.* density for both ANN and SNN to <5%, without harming event-based optical flow prediction accuracy. It is efficient regarding training efforts, requiring only single-stage training and one additional hyperparameter.
- To pursue a fair comparison, an ANN and an SNN with very similar architectures, numbers of parameters, and *ac./sp.* density are implemented onboard the same multi-core neuromorphic processor, SENECA (Tang, Vadivel et al., 2023; Yousefzadeh et al., 2022).
- We discover the different spatial distributions of ANN activations and SNN spikes. The fact that the SNN produces fewer pixels with spikes results in fewer memory accesses and thus higher time and energy efficiency.

2. Related works

2.1. ANN vs. SNN comparison

Some key characteristics of ANN *vs.* SNN comparison works are shown in Table 1. From left to right, the items are: whether multiple forward-passes were required for an SNN prediction; how the SNN(s) was trained; whether the energy cost of memory access (reading weights, reading and writing neuron states) was considered; what techniques were adopted to optimize ANN inference; what processor was adopted for neural network inference; the maximum average number of spikes received per synapse for the SNN(s) to be more energy efficient than its ANN counterpart(s). The meaning of other items in the table are introduced as follows: "✔&✖" means the studied SNNs have different results; "–" means the item was not clearly stated; "✔" or "✖" following the maximum spikes data indicates whether the studied SNNs are more energy-efficient; "ANN-SNN Conversion" means the SNN(s) was converted from trained ANN(s); "Acti. Spars". indicates that the activation sparsity of ANNs was exploited to reduce energy cost. When an activation is zero, all computations and memory accesses needed for the synaptic operation can be saved. The "*" on our work means that we not only exploit but also increase the *ac./sp.* sparsity.

It was shown in Dampfhoffer et al. (2022) that memory access is much more expensive than computation (>95% of total energy cost), thus using AC instead of MAC has a small impact on the overall energy cost. About the processors in Lemaire et al. (2022), 4 accelerators for sequential and paralleled processing are synthesized on FPGA. However, the SNN accelerator does not support parallel processing for convolution layers while the ANN accelerator does, harming the fairness of comparison. The highlight of Deng et al. (2020) is that it is the only work that studied tasks of event cameras. It was found that SNNs are less accurate and require more computation when the inputs are images and SNNs require multiple forward-passes. On the contrary, when the inputs are from event cameras and SNNs do a single forward-pass, SNNs are better in both accuracy and computation cost. The shortcoming is that only the number of operations was shown. Similarly, our work studied an event-based vision task where an SNN achieves better efficiency but, compared with Deng et al. (2020), we show more comprehensive experimental results based on accelerating the inference of both ANN and SNN.

2.2. Activation sparsification

Exploiting and increasing ANN activation sparsity is mainly motivated by improving the network inference speed on CPUs. Authors of Yang et al. (2021) proposed to rank the channels of a feature map based on their activation magnitudes. Only channels with top rankings will participate in the computations of the following layers. Similarly, for the outputs of multi-layer perceptrons, all other than the largest k (hyperparameter) entries are set to zero in Li et al. (2023). It was found that inducing activation sparsity improved training robustness to label noise and image noise, and led to better prediction confidence. Instead of sorting activations by magnitudes, the authors of Cao et al. (2019) proposed to skip the computations that are predicted to produce zero activations by a quantized version of the same network layer. Note that the works mentioned above induced computational overhead to exploit activation sparsity (Cao et al., 2019; Li et al., 2023; Yang et al., 2021). To avoid the overhead, our solution directly ignores activations smaller than the trainable activation thresholds that stay constant during inference.

Table 1
Related works on ANN *vs.* SNN comparison.

Work	Multi. pass	SNN training	Memory energy	ANN optimization	Processor(s)	Maximum spikes
Deng et al. (2020)	✔& ✖	Direct Train Wu et al. (2019)	✖	–	–	–, –
Davidson and Furber (2021)	✔	–	✔	–	SpiNNaker Rhodes et al. (2018)	1.72, –
Lemaire et al. (2022)	✔	ANN-SNN Conversion	✖	–	2 for ANN, 2 for SNN.	4.0, ✔& ✖
Lee et al. (2021)	✔	ANN-SNN Conversion	✔	Acti. Spars.	SCNN Parashar et al. (2017)	–, ✖
Dampfhoffer et al. (2022)	✔	ANN-SNN Conversion	✔	Acti. Spars. & Data Reuse	Eyeriss v2 Chen, Yang, Emer, and Sze (2019)	0.37, ✖
Ours	✖	Direct Train Wu et al. (2019), Hagenaars et al. (2021)	✔	Acti. Spars.* & Data Reuse	SENECA Xu et al. (2024)	–, ✔

As for increasing ANN activation sparsity, the training scheme proposed by Grimaldi et al. (2023) produces a set of individual pixels that are forced to have zero activations on all of their channels, leading to columns of the activation matrix skipped when using the im2col-based general matrix multiply (GEMM) technique and thus accelerates computation. A $L1$ loss that regularizes the activation map was adopted to punish big activation values and thus produce more zero and near-zero activations (Georgiadis, 2019). After fine-tuning with this regularizer, activation density was decreased to ~50% of the original model and the accuracy slightly increased. Similarly, authors of Runwal et al. (2023) used the hyperbolic tangent (TanH) function with a scaling parameter to sparsify the activation maps of rectified linear unit (ReLU) and a differentiable approximation of the $L0$ norm for other activation functions. In Kurtz et al. (2020), Hoyer regularization (Hoyer, 2004) was applied to activations of a pre-trained network during fine-tuning. Besides, the forced activation threshold ReLU activation function (FATReLU) (Eq. (1)) was proposed to replace ReLU. Notably, FATReLU is not differentiable. Therefore, Kurtz et al. (2020) performed a manual binary search of desired FATReLU thresholds based on whether the network accuracy can be recovered by retraining after applying the thresholds. Significant GPU hours and human labor are required by the manual search, especially for deep networks with many layers. To avoid such big workloads, we propose an approach (Section 4.1) that makes the FATReLU thresholds trainable, requiring only a single training attempt.

3. Networks and hardware implementation

3.1. FireNet architecture

The network architecture we study is shown in Fig. 1. It is based on the FireNet in Hagenaars et al. (2021). The ANN version and the SNN version have the same architecture of six convolution (*conv*) layers and two recurrent convolution (*rnn-conv*) blocks. All *conv* layers are single-strided and have 3-by-3 kernels except the last layer for flow prediction, which has 1-by-1 kernels. All intermediate tensors have 32 channels.

For ANN, a *conv* layer has non-zero trainable biases. To have a fairer comparison with the SNN, we made two adaptations from the RNN-FireNet (with vanilla ConvRNN) in Hagenaars et al. (2021). Firstly, we use FATReLU (Kurtz et al., 2020) activation function (to be introduced in Section 4.1) instead of ReLU to pursue higher activation sparsity. Secondly, we modified the vanilla ConvRNN block used by Hagenaars et al. (2021), shown in Appendix B. As shown in Fig. 1, our *rnn-conv* block (deployed in SENECA Core 2 and 5) has two *conv* layers instead of three, and the TanH activation function is replaced by FATReLU. This modification reduces the model size and results in sparse information instead of dense one remembered within the *rnn-conv* block. The computational demands for the recurrent *conv* layer (shown below the Tensor Sum icon) and the following layer are greatly reduced thanks to the sparse hidden state. Moreover, as to be shown in Section 5.2, our smaller-sized and sparser FireNet models have higher accuracy than the original versions reported in Hagenaars et al. (2021). The SNN FireNet in this work is the same as the LIF-FireNet in Hagenaars et al. (2021), all its *conv* layers have zero biases.

The presentation of the event camera measurements consists of per-pixel and per-polarity event counts, same as Hagenaars et al. (2021). It gets populated with consecutive, non-overlapping partitions of the event stream. For training, each input partition (referred to as an event frame) contains a fixed number of events (1000). For testing, events within a certain time duration are accumulated in an event frame.

3.2. Processing mechanism of SENECA

SENECA is a programmable digital neuromorphic processor designed with a scalable number of cores (Tang, Vadivel et al., 2023; Yousefzadeh et al., 2022). Each SENECA core consists of a high-bandwidth SRAM data memory. In this work, we use two sizes for data memory, 256 Kilobytes and 2 Megabytes, for different image resolutions, 56×56 pixels and 120×120 pixels, respectively. The resolution is constrained by the size of data memory due to the need to store the neuron states. A SENECA core has 8 neuron processing elements (NPEs) that operate in a vector-like fashion. The NPEs are hardware functional units that are time-multiplexed to perform neuron activity computations, *e.g.* addition, multiplication, comparison of numerical values, read from memory, write to memory, *etc*. A SENECA core has a RISC-V controller to conduct other operations, *e.g.* decoding received *event*,[3] calculating the addresses of the neuron states and weights, encoding *ac./sp.* into events, *etc*. More details can be found in Appendix C, D and Tang, Vadivel et al. (2023) and Xu et al. (2024).

[3] In the context of communication between SENECA cores, an *event* refers to the inter-core message. An *event* has 32 bits, In this work, the directions of *event* flows are shown in Fig. 1. To distinguish, we refer to the measurements of event cameras as *camera spikes*.

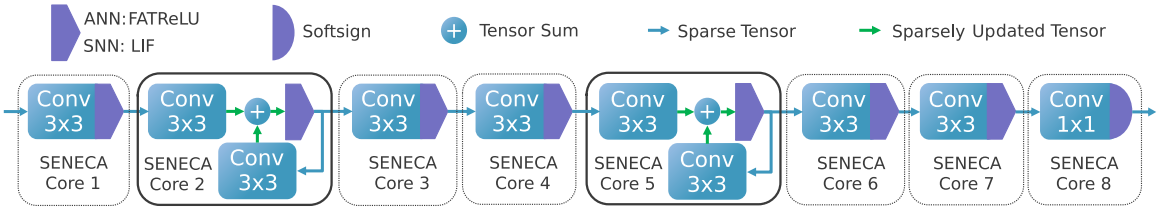


Fig. 1. FireNet network architecture used in this work for event-based optical flow prediction. As stated at the top left of this figure, for ANN, the output tensors of *conv* layers are input to the FATReLU activation function. For SNN, the spike tensors are integrated into the membrane potential of the LIF neurons as synaptic input currents. When the input tensor of a *conv* layer is sparse (blue arrow), a considerable number of pixel locations of the output tensor are not updated by the input tensor. A green arrow indicate such a sparsely updated tensor. In Hagenaars et al. (2021), the flow prediction layer has the TanH activation function. We replace it with Softsign due to its more efficient implementation on SENECA.

We map a *conv* layer or a *rnn-conv* block to a SENECA core, as shown in Fig. 1. Eight SENECA cores are cascadedly connected. In the processing of the first *conv* layer, the input *camera spikes* are pre-sorted in spatial order. *Camera spikes* that are closer to the upper edge of the frame are received earlier. For *camera spikes* on the same row, the ones closer to the left are received earlier. Once *camera spike(s)* at a pixel is received, the neuron states within the patch of pixels whose receptive fields cover the *camera spike*'s pixel location are located and updated accordingly. *Conv* layers of FireNet have 3-by-3 kernels, so a spike updates $3 \times 3 \times 32$ neurons in a 3-by-3 pixel patch. When the newly coming spike is on a different pixel, the program switches to the new set of neuron states according to the new pixel location. Because spikes come in spatial order, it is deterministic whether there will be possible future spikes falling in the receptive field of neurons at a pixel location. If not, it means that a pixel location is fully updated and thus the neuron states at this location can be input to FATReLU (for ANN) or compared with the firing threshold (for SNN). SNN spikes and ANN activations bigger than FATReLU thresholds will then be sent to the following layer in another SENECA core by SENECA *events*. Smaller ANN activations are ignored. Thus SENECA exploits ANN's activation sparsity. Since the neurons are fired in spatial order, the *ac./sp.* sent to the following layer are in the same spatial order. The following layer in another core starts processing as soon as receiving the first *ac./sp.*. At the same time, the processing of the first layer is still ongoing, leading to parallelization of the layers/cores that requires less network inference time cost than single-thread processing. This processing mechanism is called event-driven depth-first convolution. It is illustrated in detail in Appendix D.

The *rnn-conv* block's two *conv* layers are processed one after another. The processing of the recurrent layer (shown below the Tensor Sum icon in Fig. 1) starts after finishing the forward layer on the primary data pass (shown on the left the Tensor Sum icon in Fig. 1). It is possible to map the recurrent layer in another core to parallel the two *conv* layers of the *rnn-conv* block. The latency (required time to get all optical flow predictions) remains the same but it could reduce the time cost of the *rnn-conv* block to finish all processing. We leave it to future works. To reduce the number of memory-accessing operations, it is natural to perform data reuse. SENECA reuses neuron states by a mechanism called *ac./sp.* grouping. *Ac./sp.* at the same pixel location can be processed together because, in convolutional operation, they update the same set of neurons. We set the group size to four. Firstly, the neuron states are loaded from the memory. Then, four *ac./sp.* are integrated into the neuron states. Lastly, the neuron states are stored back in memory. In this way, the number of memory access operations required to process four events is reduced from four to one. If there are fewer than four *ac./sp.* at a pixel location, dummy zero *ac./sp.* fill(s) the group. In addition to neuron states, an *ac./sp.* is used by all 8 NPEs to multiply with 8 weight parameters and then update 8 neurons in 8 channels of a pixel location. Weights are not reused by our processing mechanism.

4. Activation sparsification methodology

In this section, we describe the approaches to obtain higher *ac./sp.* sparsity for both types of networks. Note that Sections 4.1 and 4.3 apply to ANN. Section 4.2 applies to both ANN and SNN.

4.1. Trainable thresholds for activations (ANN)

This subsection introduces the activation function of the ANN. A ReLU activation function sets all negative activations to zeros and reserves positive activations. As a variant of ReLU, forced activation threshold ReLU (FATReLU) proposed in Kurtz et al. (2020) further sets positive activations that are smaller than the threshold T to zeros. By adjusting the threshold values, it is possible to reach higher activation sparsities while preserving the network prediction accuracy. FATReLU's mathematical expression is

$$FATReLU_T(x) = x \cdot BS_T(x),$$
$$BS_T(x) = \begin{cases} 1 & if \quad x > T \quad (T > 0), \\ 0 & otherwise. \end{cases} \tag{1}$$

$FATReLU_T(x)$ is discontinuous and not differentiable when $x = T$. So, unlike ReLU, a network with FATReLU cannot be directly trained. To avoid the big workload of manual search of T (Kurtz et al., 2020) as introduced in Section 2.2, we borrow the idea of surrogate gradient from SNN training. As shown by Eq. (1), FATReLU is the multiplication of the input activation x and a binary step function BS_T with threshold T. BS_T is the same as the spiking function of SNN. We select the derivative of the inverse tangent as the surrogate gradient function for BS_T, same as Hagenaars et al. (2021). In this way, T becomes a trainable network parameter. We do channel-wise FATReLU thresholding, which means that there are in total 7×32 FATReLU threshold parameters in FireNet. All thresholds are initialized to a small value (1e−6) before training.

4.2. Sparsification regularizers (ANN and SNN)

From Eq. (1), we can notice that smaller FATReLU inputs x and bigger FATReLU thresholds T can increase the number of zeros. Similarly, smaller neuron membranes and bigger firing thresholds can lead to fewer spikes of SNN. Therefore, for both ANN and SNN, we add two sparsification regularizers to punish big neuron membranes and encourage thresholds T to grow, respectively. We use $L1$ loss for neuron membrane and $L2$ loss for T. The total training loss is

$$L = L_{flow} + \lambda_s \cdot L_s,$$
$$L_s = \sum_i \lambda_i \cdot \left(\sum \mathrm{ReLU}\left(\mathbf{x}_i\right) + \sum_j \left(\frac{1}{T_{i,j}} \right)^2 \right), \tag{2}$$

where L_s is the loss made of the two activation sparsification regularizers. i is the layer index, and j is the channel index. $\mathbf{x}_i$ is the neuron membrane tensor of layer i. $\sum \mathrm{ReLU}(\mathbf{x}_i)$ is the summation of all the elements of $\mathrm{ReLU}(\mathbf{x}_i)$. $T_{i,j}$ is the FATReLU threshold of ANN or the firing threshold of a LIF neuron.

Table 2
Accuracy indicated by average endpoint error (AEE) and *ac./sp.* density (Dens.(%)) of the networks tested on the MVSEC (Zhu et al., 2018) dataset (dt = 4). $\%_{\text{Out.}}$ is the percentage of outlier optical flow predictions. "-S" means that the network is trained with the activation sparsification loss (Eq. (2)). "-S*" means that the network has FATReLU with trainable thresholds but it is trained with the loss $L_{flow} + \sum_i \lambda_i \cdot (\sum \text{ReLU}(\mathbf{x}_i))$, without the regularizer for bigger FATReLU thresholds. "-FT" means that the network is trained by fine-tuning a trained model as described in Section 4.3. Other networks without "-FT" are trained from scratch. For all metrics, lower is better. The best of each metric is marked by **bold** text.

Network	outdoor_day1		indoor_flying1		indoor_flying2		indoor_flying3		Average		
	AEE	$\%_{\text{Out.}}$	AEE	$\%_{\text{Out.}}$	AEE	$\%_{\text{Out.}}$	AEE	$\%_{\text{Out.}}$	AEE	$\%_{\text{Out.}}$	Dens. (%)
EV-FlowNet(GRU) (Hagenaars et al., 2021)	**1.69**	12.50	2.16	21.51	3.90	40.72	3.00	29.60	2.94	29.35	–
RNN-EV-FlowNet	**1.69**	12.96	**2.02**	**18.74**	3.84	38.17	2.97	**27.91**	2.88	**27.32**	16.90
RNN-EV-FlowNet-S (smaller λ_s)	1.92	17.34	2.06	18.83	**3.56**	**37.02**	**2.88**	28.94	**2.79**	27.76	5.35
RNN-EV-FlowNet-S (bigger λ_s)	1.73	**12.20**	2.03	19.03	3.83	39.70	3.02	30.58	2.90	28.71	4.78
LIF-EV-FlowNet	1.99	15.99	2.47	26.79	4.94	50.51	3.91	39.59	3.68	37.47	9.46
LIF-EV-FlowNet-S (smaller λ_s)	2.01	16.88	2.69	32.00	4.77	51.29	3.84	41.85	3.66	39.90	5.81
LIF-EV-FlowNet-S (bigger λ_s)	1.88	16.36	2.76	33.63	4.96	52.65	4.06	44.76	3.80	41.68	**3.93**
FireNet(GRU) (Hagenaars et al., 2021)	2.04	20.93	3.35	42.50	5.71	61.03	4.68	53.42	4.41	49.92	–
RNN-FireNet	1.94	17.80	3.11	38.79	5.45	57.31	4.47	49.59	4.19	46.22	34.03
RNN-FireNet-S* (no threshold regularizer)	**1.67**	**12.88**	**2.79**	**32.70**	**5.02**	**51.99**	**4.05**	**43.69**	**3.80**	**40.54**	16.57
RNN-FireNet-S	2.16	22.04	3.16	40.09	5.14	55.96	4.24	48.76	4.05	46.25	5.92
RNN-FireNet-S-FT	1.97	18.31	3.24	39.23	5.48	57.00	4.45	49.02	4.22	46.09	**4.52**
LIF-FireNet	1.96	15.82	3.32	41.37	5.99	62.24	4.98	54.63	4.58	49.94	19.54
LIF-FireNet-S	2.15	21.06	3.14	39.17	5.59	57.94	4.63	51.19	4.31	47.36	4.53

4.3. Threshold initialization based on prior activations (ANN)

Using the surrogate gradient for FATReLU thresholds and loss function shown in Eq. (2), an activation-sparse ANN can be trained from scratch by a single training attempt. In practice, we find that the initialization of FATReLU thresholds affects the network performance. Thus, we propose initializing the FATReLU thresholds based on the activation statistic of a trained network. Specifically, all thresholds have fixed zero values in the first training attempt, and the network is trained from scratch by $L_{flow} + \sum_i \lambda_i \cdot (\sum \text{ReLU}(\mathbf{x}_i))$. Then, we run the trained network (inference only) on a subset of the training set to log the activation tensors. Based on the logged tensors, the initial threshold $T_{N,M,init.}$ for the Mth channel of the Nth layer is set as the median of all logged activations at the Mth channel of the Nth layer, *i.e.*, the network is initialized to have 50% non-zero activations of the originally trained network, statistically. The second training attempt fine-tunes the trained network with the initialized thresholds by Eq. (2).

5. Evaluation of activation sparsification

5.1. Network training

We train our optical flow FireNet based on the open-sourced code of Hagenaars et al. (2021). The regularizers for activation sparsification are added to the event deblurring loss for self-supervised learning of optical flow L_{flow}, as shown in Eq. (2). A reader who is interested in L_{flow} and the training strategies can refer to Hagenaars et al. (2021). The network is trained for 100 epochs on the UZH-FPV dataset (Delmerico, Cieslewski, Rebecq, Faessler, & Scaramuzza, 2019). We use the AdamW optimizer with a weight decay of 0.01 and the OneCycleLR scheduler with a maximum learning rate of 2e−4. In our practice, networks are trained with different λ_s. It ranges from 1.0e−6 to 2.6e−6 for ANN and 1.6e−7 to 2.6e−7 for SNN. As for λ_i, the layer with the highest *ac./sp.* density has $\lambda_i = 2.0$ while other layers have $\lambda_i = 1.0$. The purpose is to reduce the difference in densities between layers to avoid one or several layer(s) becoming much denser than others and becoming bottleneck(s) of inference speed.

5.2. Network accuracy and activation density

In this work, we focus on the lightweight FireNet architecture. But to show that the proposed activation sparsification approach can generalize, we also applied it to EV-FlowNet, using the code from Hagenaars et al. (2021). Note that, we do not intend to deploy EV-FlowNet on the neuromorphic processor due to its large size. So we do not modify the architecture of its recurrent block. In Table 2, we took two networks, EV-FlowNet (GRU) and FireNet (GRU), from Hagenaars et al. (2021) as the baselines of network prediction accuracy. The other networks in the table are trained by us. RNN-FireNet has 74,722 parameters. RNN-FireNet-S and RNN-FireNet-S-FT have 74,946 parameters. LIF-FireNet and LIF-FireNet-S has 74,818 parameters. RNN-EV-FlowNet has 23,531,752 parameters. RNN-EV-FlowNet-S has 23,535,240 parameters. LIF-EV-FlowNet and LIF-EV-FlowNet-S have 20,400,840 parameters. Surprisingly, our networks with vanilla RNN have better accuracy than the baselines with GRU. We attribute it to our training scheme with the AdamW optimizer and the OneCycleLR learning rate scheduler.

Ac./sp. density is calculated for the output *ac./sp.* tensors of the input layer and hidden layers. The *ac./sp.* The density of each layer is logged for all testing samples. The average *ac./sp.* density of a network model is the average over all its layers and all testing samples. The output layer has a dense activation function, SoftSign for our adapted FireNet and Tanh for EV-FlowNet. The RNN blocks of EV-FlowNet have a dense Tanh activation function. We do not apply activation sparsification to such layers and do not take the 100% activation density into account when calculating the average activation density. The density of input *camera spikes* is not taken into account either, since it depends on the dataset and thus is not correlated to the network models.

In Table 2, we show two models of the sparsified ANN EV-FlowNet with vanilla RNN (RNN-EV-FlowNet-S) and the sparsified SNN LIF-EV-FlowNet (LIF-EV-FlowNet-S), respectively. "Smaller λ_s" and "bigger λ_s" mark two separate training attempts of the same network with the same *ac./sp.* sparsification approach but different weights for sparsification loss (λ_s). The accuracy and *ac./sp.* density of the two network models tell that *ac./sp.* sparsification with smaller λ_s produces similar accuracy but higher *ac./sp.* sparsity than the network model without sparsification. *Ac./sp.* sparsification with bigger λ_s can train network models with slightly lower accuracy but higher *ac./sp.* sparsity. Accuracy-sparsity trade-off is discussed in the next subsection.

As shown in the bottom half of Table 2, RNN-FireNet-S* has the highest accuracy. It is close to the accuracy of LIF-EV-FlowNet-S which has 20,400,840 parameters, ~272.2 times of RNN-FireNet-S (74,946 parameters). The activation density of RNN-FireNet-S* is around half of RNN-FireNet. It indicates that lower density could exist together with higher accuracy. This phenomenon was also observed by works (Georgiadis, 2019; Hoefler, Alistarh, Ben-Nun, Dryden, & Peste, 2021; Kurtz et al., 2020; Runwal et al., 2023; Yang et al., 2021). We intuitively

explain this phenomenon by that the sparsification regularizers can suppress the noise in activations. Only the activations carrying important information are kept. The "denoised" sparse activations can contribute to accuracy. RNN-FireNet-S-FT has only ~13.3% activations of RNN-FireNet while maintaining a similar accuracy. For LIF-FireNet-S, we observe a small improvement in accuracy and 23.2% density of the LIF-FireNet trained without sparsification. In Section 6, RNN-FireNet-S-FT and LIF-FireNet-S are compared with hardware in the loop.

5.3. Ablation study

To better understand the effects of the sparsification regularizers, we perform an ablation study. For simplicity, we use neuron density and pixel density to refer to neuron-wise *ac./sp.* density and pixel-wise *ac./sp.* density, respectively. Pixel density is the percentage of pixel locations that have at least one channel with *ac./sp.*. For instance, an activation tensor of a *conv* layer has 5×5 pixel locations and 8 channels on each pixel. If the activation tensor has 16 non-zero elements, then the neuron-wise activation density is $16/(5 \times 5 \times 8) = 8\%$. If these 16 non-zero elements are located on 2 pixel locations (all 8 channels are non-zero for these 2 pixels), then the pixel-wise activation density is $2/(5 \times 5) = 8\%$. If there are 16 pixels each of which has only one non-zero channel, then there are 16 activated pixels and the pixel density is $16/(5 \times 5) = 64\%$. Therefore, pixel density reflects the spatial density of *ac./sp.* in the 2-dimensional domain of pixel locations, *i.e.* image plane.

The neuron density and average endpoint errors (AEEs) of network models trained with different settings of sparsification loss are shown in the left subplot of Fig. 2. For simplicity, *ANN* represents RNN-FireNet, and *LIF* represents LIF-FireNet. To remove as much of the effect of randomness in network training as possible, we trained ~10 models for each loss setting with different weights on the sparsification loss (λ_s in Eq. (2)). In the left subplot of Fig. 2, if the point of *Model A* is on the lower left side of the point of *Model B*, it means *Model A* is better in both accuracy and neuron sparsity. In this case, if the two models are trained with the same loss setting, *Model B* is omitted in both subplots.

First, we discuss the information in the left subplot. For ANN, when applying the $L1$ regularizer on neuron membrane voltage alone (*ANN-S(vol.,ReLU)*, green inverted triangle), the accuracy significantly increases and the neuron density decreases to around 50% of the networks trained without sparsification loss (*ANN (ReLU)*, orange circle). Comparing *ANN-S(vol.,FATReLU)* marked by the red crosses and *ANN-S(vol.,ReLU)*, we can notice that having FATReLU with trainable thresholds brings small improvement to accuracy and the density is little affected. After applying the $L2$ regularizer that encourages bigger thresholds, density dramatically drops to ~5%, as shown by *ANN-S(vol.&thre.,FATReLU)* marked by the green diamonds. Comparing the green diamonds with the red squares (*ANN-S-FT*), we can see that the latter performs better in both metrics, to a small extent. Similarly, for SNNs with LIF neurons, applying sparsification loss noticeably benefits both density and accuracy. A blue pentagon (*LIF-S(vol.)*) and an orange triangle (*LIF-S(vol.&thre.)*) are closely located, indicating the small effect from the $L2$ loss for the SNN firing thresholds. Then, we move to the right subplot of Fig. 2 to analyze the pixel density. As shown, SNNs have lower pixel densities (~39%) than ANNs. ANNs with high accuracy (AEE < 4.0) have high pixel density (>68%).

To summarize,

- applying sparsification loss significantly reduces *ac./sp.* density without deteriorating accuracy;
- Sparsification can dramatically increase ANNs' accuracy with only ~50% activations as the ANN without sparsification;
- For highly sparsified networks whose neuron density is lower than 6%, a negative correlation exists between accuracy and sparsity;
- SNN models are generally less accurate than ANNs but have much lower pixel density.

6. Experiments onboard neuromorphic processor

We select an ANN (RNN-FireNet-S-FT in Table 2) and an SNN (LIF-FireNet-S) to deploy onboard SENECA and conduct comparisons. We measure[4] the time and energy cost of network inference. We test on several representative input event frames instead of the whole dataset because hardware simulation is very time-consuming. It takes around 30 h to simulate 70 ms of FireNet inference.

6.1. Single layer comparisons in controlled conditions

Before testing the whole network on real data of the test set, we conduct a comparison to study how pixel density and number of *ac./sp.* on the same pixel affect the processing cost of a single *conv* layer. The results are shown in Fig. 3. In these experiments, the spatial resolution is 16×16, 256 pixels in total. Each pixel has 32 channels, the same as FireNet. We consider the total processing cost of receiving SENECA events, updating neuron states, and comparing the neuron states with the thresholds. Capturing, encoding, and sending the *ac./sp.* are disabled because it is hard to control the number of output *ac./sp.*. In the first set of experiments (top row of Fig. 3), we generate one *ac./sp.* for each pixel because most of the output tensors of FireNet layers have a single *ac./sp.* per pixel, as shown in Fig. 4. The numbers of pixels that have an *ac./sp.* are 0, 50, 102, 151, and 204. Shown in the top two subplots of Fig. 3, there is a clear linear correlation between the number of pixels having *ac./sp.* and the inference time. SNN is more time-consuming than ANN because the decoding schemes of SENECA *events* are different between SNN and ANN.

6.1.1. Different event encoding schemes for ANN and SNN

For ANN, we use a 32-bit *event* to encode an activation, 16 bits for the channel index (unsigned integer), and the rest 16 bits for the activation value (BFloat16). In contrast, for SNN, we use a bit coding scheme. The 32 bits of an *event* encode the 32 channels. If a channel has a spike, the corresponding bit is 1 otherwise 0. While for ANN, the number of *events* is the same as the number of activations. SNN requires fewer events and thus is more efficient in terms of inter-core communication. However, to decode the spikes from the 32-bit *event*, 32 iterations are required to check every bit. When there is one *ac./sp.* at a pixel, ANN takes 2.52 µs to decode. In contrast, SNN needs 5.14 µs. When the number is 28, SNN takes less time (8.04 *vs.* 8.14). This phenomenon tells us that the software implementation onboard SENECA should take into account the spike distribution of the certain network to select the better *event* encoding scheme. In this work, all experiments of SNN are conducted with the bit coding scheme. To eliminate the effects of *event* encoding schemes, we subtract the time difference for the SNN assuming it uses the same encoding as ANN, shown by gray circles in the top row of Fig. 3. After the subtraction, the SNN is more efficient. We attribute it to the fact that a binary spike does not require the memory access and multiplication required by an activation. Note that we use the power measurements of the SNN with the bit encoding scheme when calculating the energy cost. Using ANN's encoding scheme should lead to lower power cost and thus less energy than what the top-right subplot of Fig. 3 shows.

[4] All hardware-related measurements were performed in gate-level simulation using industry-standard ASIC simulation and power measurement tools (Cadence Xcelium and Cadence JOULES) for GF-22 nm FDX technology node (in the typical corner 0.8 V and 25C, no back-biasing, 500MHz clock frequency). The power results are accurate within 15% of signoff power and include the total power consumption of the chip, i.e. both dynamic and static power. We have not included the I/O power consumption in the reported results.

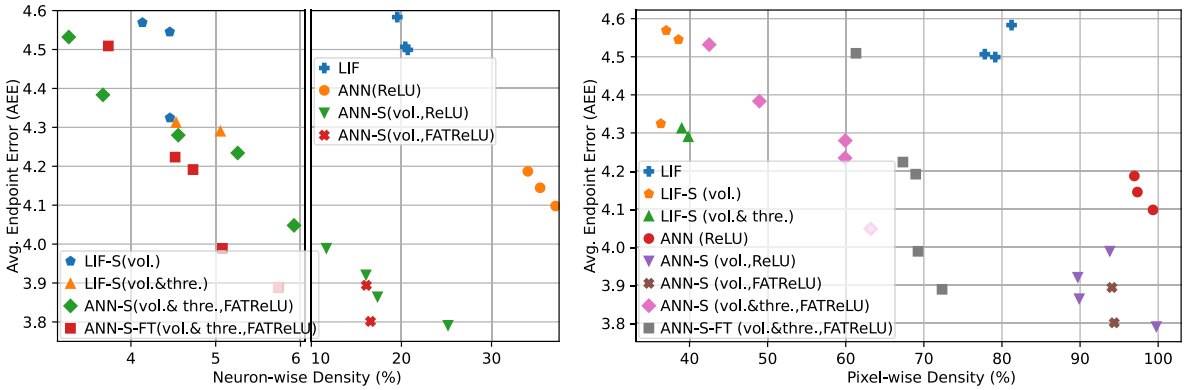


Fig. 2. The correlation between prediction accuracy and *ac./sp.* density. The accuracy metric is the AEE over the whole test set. The shown neuron-wise and pixel-wise density (x-axis) is the average of all the layers and all testing samples. The networks shown in both subplots are the same. "*vol.*" in the legend means that the networks are trained with the $L1$ regularizer for neuron membrane voltage. "*vol.&thre.*" means the sparsification loss involves both the $L1$ voltage regularizer and the $L2$ regularizer that encourages the ANN's FATReLU thresholds or SNN's firing thresholds to grow. There is no network whose neuron density is between 6% and 10% so the x-axis of the left subplot is truncated.

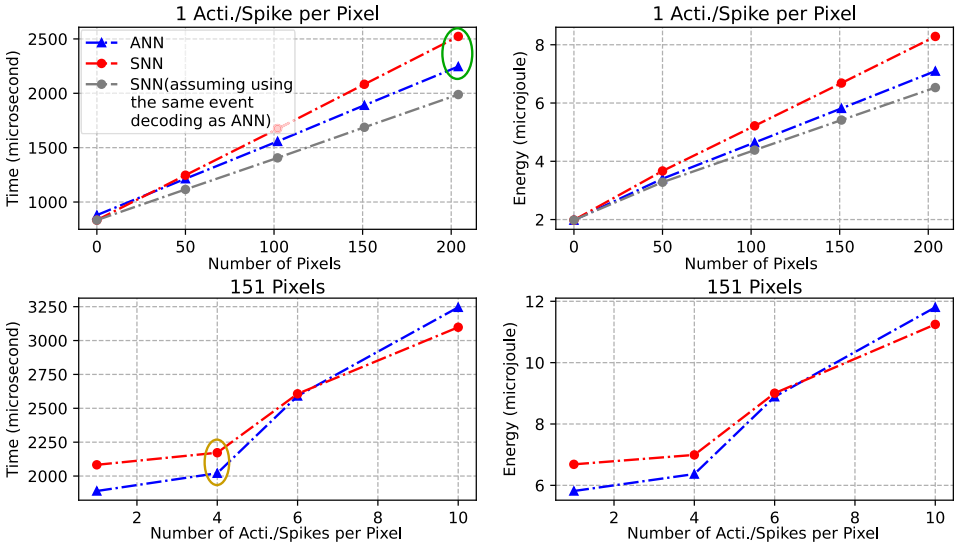


Fig. 3. Time and energy cost of the controlled experiments. In the bottom two subplots, 151 pixels have at least one *ac./sp.*.

6.1.2. *Effect of ac./sp. grouping strategy on time and energy cost*

As shown in the two bottom subplots of Fig. 3, increasing the number of *ac./sp.* from 1 to 4 leads to less growth in time and energy cost than increasing it from 4 to 6. It is because of the *ac./sp.* grouping strategy that processes four spikes together, introduced in Section 3.2. When the number is below 5, it requires only one neuron state updating operation. When the number increases to 6, it requires two operations thus the cost increases significantly. When the number is 10, three operations are required. Comparing the data points in the circles in the two subplots on the left, the points in the green circle have 204 *ac./sp.*. The points in the yellow circle have 604. Although having more *ac./sp.*, the time costs of the points in the yellow circle are less, due to the *ac./sp.* grouping mechanism that reduces the required numbers of memory access operations of neuron states. Therefore, higher neuron density does not necessarily mean more time cost, the spatial distribution of *ac./sp.* is an important factor. SENECA processing can be more efficient when the *ac./sp.* are concentrated at fewer pixels.

6.2. *Spatial distributions of ac./sp.*

After the experiments of *ac./sp.* distribution of a single layer in controlled conditions, we compare and analyze the *ac./sp.* distributions of the ANN and the SNN on the real test set, and then deduce which network type can be more efficient. The boxplots reflecting the distribution of pixel density are shown in the left subplot of Fig. 4. ANN has a higher pixel density than SNN in 6 out of 7 layers. For the last 3 layers, almost all inferences have almost 100% pixel density. In the right subplot of Fig. 4, we can see that the SNN statistically has more spikes per pixel than the ANN. It is reasonable because the neuron densities of the ANN and SNN are similar and the SNN has lower pixel density. Based on Fig. 4, we claim that the SNN is more likely to have higher efficiency because its spikes are concentrated at fewer pixels. As discussed in Section 6.1, this feature benefits processing efficiency.

6.2.1. *Layer-wise visualization of ac./sp.*

Fig. 5 visualizes the accumulated *camera spikes* (*i.e.* event frame that is the network input), *ac./sp.* maps produced by each layer, and event-based optical flow predictions. The caption for Fig. 5 introduced each type of its subfigure in detail. From Fig. 5, we notice that ANN has higher pixel-wise activation density, especially for Layers 5, 6, and 7. Almost every pixel has at least one activation while the number of activation(s) at one pixel is small, as indicated by the dark gray color in the first and third rows of Fig. 5 that correspond to the ANN's activation maps. In contrast, Layers 5, 6, and 7 of the SNN produced

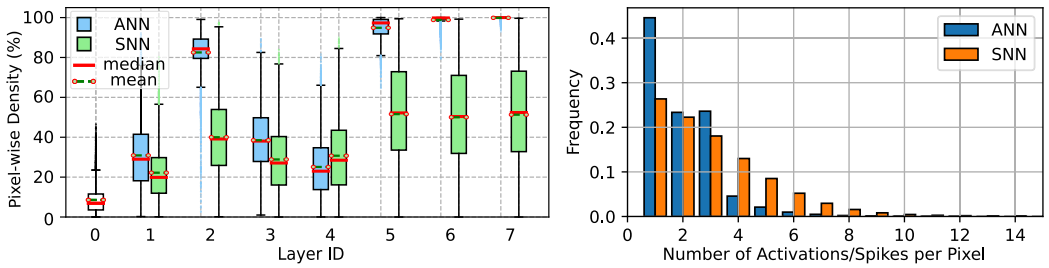


Fig. 4. Layer-wise distribution of pixel density (left) and distribution of the numbers of *ac./sp.* per pixel (right) based on the data logged in testing.

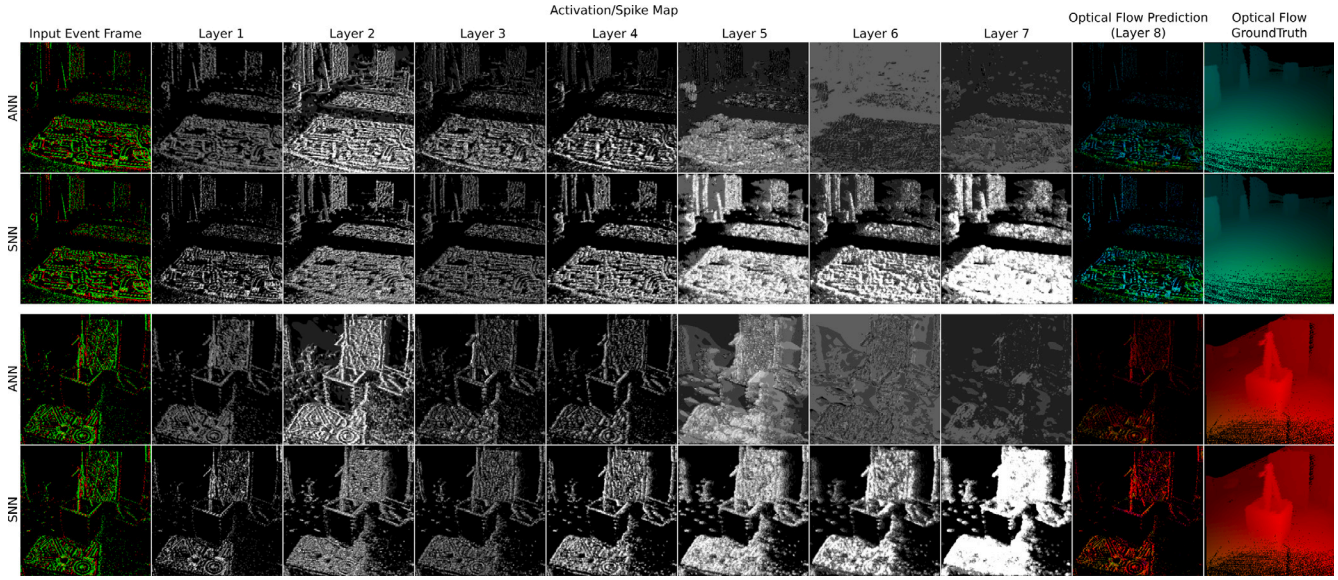


Fig. 5. Visualization of event-based optical flow prediction on two testing event frames. The first column shows the accumulated *camera spikes* (event frames) that are the input to the networks. Events within the time bin of 12.5 ms are accumulated in the image frame. Events in red or green capture brightness changes in two polarities (turn brighter and turn darker). The 2nd to 8th columns show the *ac./sp.* density of each pixel. For layers 1 to 7, their *ac./sp.* tensors have 32 channels. If there are no non-zero *ac./sp.* at a pixel, its color is black. If there are 8 or more than 8 channels that have non-zero *ac./sp.* at a pixel, its color is white. The bigger the number of *ac./sp.*, the brighter the pixel. The second column from the right shows the optical flow prediction from the network. Only pixels that have at least one input event have an optical flow prediction, which is a 2-dimensional vector in the image plane, encoded by colors as shown in Fig. 6. The column on the right shows the dense ground truth optical flow measured by other sensors, provided by the testing dataset (Zhu et al., 2018).

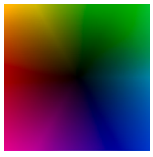


Fig. 6. Optical flow color coding scheme. Direction is encoded in color hue, and speed in color brightness.

many pixels without any spike. Most of such dark pixels are in the image regions where there is no input *camera spikes*. For image regions with many *camera spikes*, all layers of the SNN produce dense spikes. It can be summarized that higher contrast in an *ac./sp.* density map means a higher degree of spatial concentration of *ac./sp.* and higher pixel sparsity. The spike density maps of the SNN have higher contrast than the activation maps of the ANN. Thus SNN has higher pixel sparsity. This observation is consistent with Fig. 4.

6.2.2. Layer-wise ac./sp. density of a relatively complicated network architecture

Figs. 4 and 5 show the pixel-wise *ac./sp.* density of the FireNet architecture. One feature of FireNet is that the output tensors of all layers have the same spatial size, *i.e.*, there is no downsampling or upsampling. For EV-FlowNet, a more complicated UNet-like architecture with downsampling encoding and upsampling decoding, we got similar results. The RNN-EV-FlowNet-S (4th row in Table 2) has average (over the testing dataset) pixel density (%) at each layer: [100, 85.39, 100, 100, 99.99, 69.1, 71.65, 65.78, 99.99, 100, 99.98, 100]. In contrast, the layer-wise pixel density (%) of the LIF-EV-FlowNet-S (7th row in Table 2) is: [57.97, 63.69, 79.77, 90.49, 88.09, 70.05, 89.26, 85.07, 89.23, 85.25, 79.54, 67.89]. For 9 out of 12 layers, the ANN (RNN-EV-FlowNet-S) has a higher pixel density than its SNN counterpart (LIF-EV-FlowNet-S).

6.3. Network experiments

Recalling Section 3.2, we test two resolutions on two sizes of on-chip memory of SENECA. We first show and discuss the time and energy costs of the event frames with 56×56 pixels and then the results of 120×120 pixels. The network prediction accuracy of 120×120 pixels onboard SENECA follows. For each type of network, we select three event frames from the test set that lead to different neuron densities. For each layer of the SNN, the frames lead to respectively ~20%, ~100%, and ~200% of the average neuron density of the layer over the test set. For the ANN whose neuron density is narrower distributed, the three input frames respectively lead all layers except the 6th layer to have ~50%, ~100%, and ~150% of the average layer-wise neuron density. Although the 6th layer does not coincide with expectation, the two selected frames for ~50% and ~150% density have the least

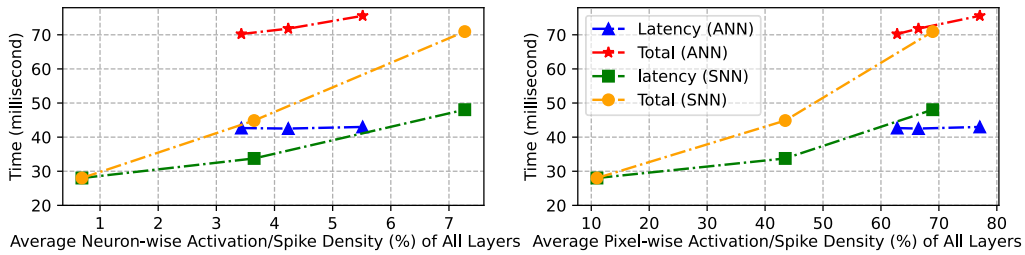


Fig. 7. Correlation between the average *ac./sp.* density of all layers and network inference time cost measured on the three testing frames with 56 × 56 pixels. The density of the input *camera spikes* is not taken into account for the average density. We define the latency of processing a frame as the duration from when the first *camera spike* is input to the network to when all optical flow vectors have been predicted. We define the total time cost as the duration from when the first *camera spike* is input to the network to when all processing including the *rnn-conv* blocks introduced in Section 3.2 has been finished. The data points in the two subplots are the results of the same testing frame if their horizontal sequence (left–right) is the same.

Table 3
Total energy cost of the Testing Frames (56 × 56 pixels).

Network and density	ANN 50%	ANN 100%	ANN 150%	SNN 20%	SNN 100%	SNN 200%
Energy Consum. (μJ)	1115.0	1233.0	1340.5	391.1	927.0	1320.0

Table 4
Accuracy and *ac./sp.* density (dt = 4) of the networks tested on downsampled event frames with 120 × 120 pixels, running on GPU or SENECA. "Dens. (%)" refers to neuron density.

Sequence	ANN				SNN			
	AEE		Dens. (%)		AEE		Dens. (%)	
	GPU	SENECA	GPU	SENECA	GPU	SENECA	GPU	SENECA
outdoor_day1	3.23	3.08	3.65	3.62	3.82	3.75	2.04	1.99
indoor_flying1	4.18	4.05	4.22	4.17	3.68	3.65	3.62	3.52
indoor_flying2	5.73	5.67	4.67	4.62	5.60	5.61	4.88	4.78
indoor_flying3	4.77	4.71	4.50	4.45	4.67	4.67	4.41	4.30
Average	4.77	4.68	4.40	4.35	4.62	4.61	4.13	4.03

differences from the expected densities among all frames in the test set.

6.3.1. Network inference time and energy costs

Fig. 7 shows the time cost and its correlations to neuron density and pixel density. Same as the results shown in Section 6.1, we see a positive correlation between pixel density and time cost. The three frames of the ANN have similar time costs because ANN's last three layers have almost 100% pixel density on almost all testing frames (shown in Fig. 4), including the selected three frames. These layers took longer time than other layers due to the high pixel density. Because of the parallelization of the layers/cores introduced in Section 3.2, those layers became the bottlenecks of the whole network inference. Given the similar pixel densities (~100%) of the bottleneck layers of the testing frames, it is reasonable to get similar time costs. As shown in the right subplot of Fig. 7, when the pixel density is ~70%, the SNN has a larger latency than the testing frames of the ANN. The reason is that the SNN testing frame has noticeably higher neuron density, as shown in the left subplot. Recalling the two bottom subplots of Fig. 3, more *ac./sp.* on the same pixel leads to a longer time.

Comparing the data points corresponding to the average neuron densities (the one in the middle for each polyline), it is noticeable that the SNN has higher time efficiency. The SNN costs 44.9 ms in processing the selected testing frame, which is 62.5% of 71.8 ms, the time cost of the ANN. The SNN testing frame for average neuron density leads to pixel density (%) [9.3, 25.2, 45.3, 30.8, 31.5, 55.4, 56.9, 59.2] for each layer, 9.3 is the density of the input *camera spikes* and the average density of the 7 layers is 43.5%. It is higher than the average pixel density (%) over the whole test set, which is [8.6, 22.2, 40.0, 28.9, 30.7, 51.6, 50.0, 51.3], 8.6 is the density of the input *camera spikes* and the average density of the 7 layers is 39.2%. Therefore, the average time cost of the SNN over the test set could be lower than the selected testing frame. As for the ANN, the testing frame for average neuron density results in pixel density (%) [8.8, 28.6, 82.5, 37.4, 22.8, 96.2, 98.3, 99.6] for each layer, 8.8 is the density of the input *camera spikes* and the average density of the 7 layers is 66.5%. It is slightly lower than the average over the whole test set, [8.6, 30.9, 82.5, 38.5, 25.1, 94.9, 98.8, 99.9], 8.6 is the density of the input *camera spikes* and the average density of the 7 layers is 67.2%. Therefore, the ANN could cost more time than the selected frame, on average over the test set. Based on the fact that the SNN's time cost is 62.5% of the ANN's when processing the selected testing frames for average neuron density, we claim that the SNN is averagely more time-efficient than the ANN on the test set and the average time cost of SNN is less than 62.5% of the ANN's. In addition, SNNs can be more time efficient if using the same SENECA *event* encoding scheme as ANN, as discussed in Section 6.1. The energy costs are shown in Table 3. The SNN's energy cost is 75.2% of the ANN's.

The time and energy costs of processing 120 × 120 pixels are listed as follows. The latency and total time of the ANN are 182.69 ms and 326.39 ms, respectively. For SNN, the measured durations are 148.91 ms and 225.77 ms, respectively. The SNN is ~18% and ~30% more efficient in latency and total time cost, respectively. The energy cost of the ANN is 5753.4 μJ and the SNN consumes 4174.4 μJ. The SNN has a ~27% advantage. In general, the advantage of the SNN over the ANN observed in the test on 120 × 120 pixels is similar to the observation in the test on 56 × 56 pixels.

6.3.2. Network prediction accuracy with the 16-bit data type onboard SENECA

The event frames with 56 × 56 pixels are downsampled from the central patch of 112 × 112 pixels. It only covers 19.14% area of the original event frame with 256 × 256 pixels. Besides, we find that, for many frames, there is no available ground truth optical flow for any pixel. Therefore, we only evaluate the network accuracy on event frames of 120 × 120 pixels on SENECA. For each of the ANN and the SNN, we select a frame producing ~100% of the average neuron density

over the whole test set on all layers. The statistical accuracy and density of the whole test set shown in Table 4 are obtained by the Python-based SENECA simulator running on a CPU server. This simulator produces exactly the same computation results as SENECA without any precision difference. SENECA uses 16-bit BFloat16 as the data type, but the network parameters trained on GPU are 32-bit single-precision floating-point numbers. We do post-training quantization that rounds the trained network parameters to BFloat16 values to deploy the networks on SENECA. We also run the same tests on GPU to compare with the results from SENECA. In general, SENECA produces slightly better accuracy and sparsity than GPU, which means that the post-training quantization to BFloat16 does not deteriorate accuracy. Intuitively, we propose the hypothesis that BFloat16 has enough precision in the context of optical flow prediction using FireNet.

7. Conclusions

In this work, we proposed an event-based optical flow solution based on *ac./sp.*-sparsified neural networks onboard a neuromorphic processor, and then conduct a fair comparison of an ANN and an SNN with the same lightweight architecture. The major findings are:

- *Ac./sp.* sparsification is proven useful for the two network architectures for event-based optical flow (FireNet and EV-FlowNet). Notably, for an ANN trained with activation sparsification, its accuracy can be greatly improved. At the same time, its neuron-wise activation density is greatly reduced;
- Given similar neuron density, lower pixel density means that, firstly, fewer neurons are required to be found (based on the *ac./sp.* pixel location), accessed, and updated. Secondly, the *ac./sp.* grouping mechanism reuses the neuron states to be updated more times because statistically there are more *ac./sp.* on a pixel. This statement can be generalized to other processing mechanisms that reuse neuron states in a *conv* layer;
- The SNN and the ANN in comparison have similar neuron density but the SNN has significantly lower pixel density, which is the main contributor to SNN's higher efficiency. It is an interesting topic to study why the ANN has higher pixel density and how to reduce it.

This work makes one step forward in the field of SNN *vs.* ANN comparison by studying a more complicated regression task of event-based vision. Although the network we deploy on the neuromorphic processor is more complicated than all the networks in the same type of literature, it is still only a lightweight network without state-of-the-art accuracy. The obstacle is our current limited capacity of mapping bigger-size networks with more complicated inter-layer connections to the multi-core processor. In the end, we hope this work can encourage the community to find more evidence showing SNN's better efficiency in fair comparisons.

CRediT authorship contribution statement

Yingfu Xu: Writing – original draft, Visualization, Validation, Software, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Guangzhi Tang:** Writing – review & editing, Methodology, Conceptualization. **Amirreza Yousefzadeh:** Writing – review & editing, Supervision, Methodology, Investigation, Funding acquisition, Formal analysis, Conceptualization. **Guido C.H.E. de Croon:** Writing – review & editing, Supervision, Methodology, Conceptualization. **Manolis Sifalakis:** Writing – review & editing, Supervision, Project administration, Methodology, Funding acquisition, Formal analysis, Conceptualization.

Acknowledgments

This work was carried out at Stichting imec Nederland, supported by the DAIS (Distributed Artificial Intelligent System) project funded by Horizon 2020 (101007273), the REBECCA (Reconfigurable Heterogeneous Highly Parallel Processing Platform for safe and secure AI) project funded by Horizon Europe (101097224) and the "ACT: Perceptive Acting Under Uncertainty" project funded by the Dutch Research Council NWO-NWA (NWA.1292.19.298).

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix A. Supplementary discussion on benchmarking ANNs and SNNs

Advances in ANN models target primarily task accuracy and computational speed (on vector accelerators *e.g.* GPUs and TPUs). For example, the transformer architecture addresses the limitations of training effective RNNs and exploits parallelization that has a computational bottleneck of RNNs. Follow-up ramifications as par example the elimination of the softMax bottleneck, or factorizations to confine the quadratic cost of self-attention, are in a similar spirit. From this point of view, ANN technology sees always a trade-off between latency and energy and until now opts mainly for the former.

SNN models on the other hand target primarily energy and latency efficiency in tandem, while trying as much as possible not to compromise the task accuracy seen in ANNs, critically taking into account the "qualities" of the type of computing substrate/hardware they are meant for (*e.g.* dataflow event-driven architectures where memory and processing are co-located or located near each other). From the efficiency point of view, algorithm-hardware co-design and co-optimization are always for the taking.

Because the primary targets or objectives of SNNs and ANNs are different, and they are pursued or achieved through different means, an upfront quantitative comparison is inherently problematic or biased the exact moment that single-value summary metrics are employed, or when they were designed for one context but applied to another where the same assumptions may not hold.

Take for example accuracy as a metric. ANNs have a long history of structural explorations and training optimizations to pursue the highest accuracy possible. This includes, among others, the way input is ingested in these models. When comparing ANNs to similar SNN model architectures derived by conversion from ANNs and ingesting similar frame-based input after conversion to rate-based spike codings, it is not difficult to imagine that, at best, they will fare similarly or close in terms of accuracy. The question is, what has this conversion amounted to in terms of efficiency, which is the primary objective of SNNs? More often than not, conversion models exhibit dense activations. Rate-based encoding from frame data is extremely dense, and ANN-inspired optimization tricks to boost accuracy such as normalization layers deteriorate sparsity. However, this mediocrity for SNNs in such studies is just the result of trying to "fit an SNN in ANNs' shoes", rather than comparing the two fairly.

Another example is taking activation sparsity as a single-number metric of efficiency. One may argue that a fair expectation is if we could trade off a (small) amount of accuracy for a good amount of efficiency. If sparsity equals efficiency, then counting the zero activations and multiplying by the unit energy of a memory transaction becomes a comparison metric. However, this is rather superficial and often does not match what one sees in an actual measurement-based comparison.

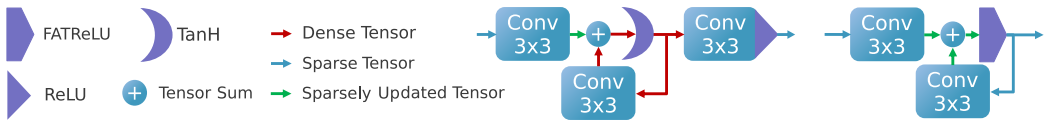


Fig. 8. The vanilla *ConvRNN* block adopted by Hagenaars et al. (2021) (left) and our recurrent block (right).

If the models are executed in GPU or other vector-based accelerators, which are mainly tailored to ANNs, sparsity that follows synaptic structure (structured sparsity) modulo-ed by the size of the vector pipelines is exploitable for saving memory transactions and computations, both in ANNs and SNNs. However, in SNNs, a large part of the activation sparsity is unstructured and temporal, which can only be exploited by event-driven and neuromorphic accelerators that are often unaccounted for in comparisons. Memory and communication bottlenecks have been almost completely overlooked in comparison studies. Dense activation propagation between layers after layer synchronization causes interleaving memory and communication bottlenecks and consequently increases the latency of ANNs. Sparse activations imply fewer memory transactions and event-driven processing implies that unstructured sparsity can be exploited for pipelining the across-layer memory transactions of activations. Sparse activations and event-driven processing alleviate interleaving bottlenecks and make better use of computational resources, appealing for algorithm-hardware co-optimization.

Overall, we defend that for a fair comparison, one should delve deeper into several of these aspects and explore when and under what conditions sparsity correlates with efficiency, as well as how accuracy correlates with sparsity and how to trade-off. The selection of suitable applications or tasks for benchmarking is not less important for an insightful study and comparison. What we see in most comparison studies are overly simple tasks and models that have become legacies over the years. Although suitable for proof-of-concept explorations, they are outdated for real-world comparative studies. On the other hand, application tasks and network models in some other comparison studies are appropriately challenging for comparative black-box benchmarking, but too complex for white-box (introspective) analyses.

To address this gap, the NeuroBench suite (Yik et al., 2023) emerged recently from a community effort, intending to define task benchmarks complementary to MLPerf (Reddi et al., 2020) and introducing a neuromorphic perspective to benchmarking. ANNs and SNNs can be compared as black boxes based on multiple metrics that are more pertinent to neuromorphic objectives. Their efficacies on hardware substrates are also assessed. For example, Neurobench considers tasks with rich temporal information and thus can give rise to temporal sparsity under event-driven processing. Neurobench considers metrics such as activation sparsity versus effective sparsity and synaptic operations, which indicate how well sparsity is exploited on the underlying hardware and contributes to latency or energy efficiency. The overarching intent is that if an SNN and an ANN are benchmarked in both NeuroBench and MLPerf, then a more objective and holistic assessment of their merits is available.

NeuroBench is a very recent development, which was not available at the time of this work. Support for different hardware is underway, and its task coverage in the current version is relatively limited, albeit it does offer regression and classification benchmarks. Additionally, like MLPerf, Neurobench enables a goal-oriented black-box comparison of ANN and SNN models, which differ not only in information encoding, but also in topological structures, parameterizations (*e.g.* synaptic delays), and others. This arguably is another dimension to the fairness of comparison that may reflect the different objectives of ANNs and SNNs.

Appendix B. ANN recurrent block

The difference between the recurrent block of FireNet we use and the original one (Hagenaars et al., 2021) is shown in Fig. 8. The original one has one more *conv* layer and uses a TanH activation function. TanH produces a non-zero output for a non-zero input. Thus, the output tensor has almost zero sparsity (red arrow). The main purpose of our modification is to get the ANN and the SNN in comparison to have the same network architecture and sparse feature maps. In terms of model capacity, our modification is disadvantageous, but our modification greatly increases activation sparsity. In addition, the accuracy of our models with sparsification-aware training does not deteriorate.

Appendix C. Event-driven neural processing on SENECA

This section was first introduced in Xu et al. (2024). We include it here and adapt the writing for the context of this work for ease of reading. An interested reader can refer to Xu et al. (2024) for more details.

SENECA is a programmable digital neuromorphic processor that is capable of performing a wide range of tasks. The processor is designed with a scalable number of cores. Each core consists of data memory, a flexible controller (RISC-V), a dedicated controller (loop controller), an event capture unit, multiple (configurable, in this work, 8) neuron processing elements (NPEs) that operate in a vector-like fashion, and a programmable Network on Chip (NoC) which facilitate the event communication among the cores. The NoC delivers a SENECA inter-core events (referred to as *event* for simplicity) to the destination core based on the content of its routing table, which can change dynamically by RISC-V. In this work, given the cascaded layer connection of FireNet, events are sent from one core to another in a single direction.

The NPEs are hardware functional units that are time-multiplexed to perform neuron activity computations. Each NPE is connected to a high-bandwidth SRAM data memory (16 bits for each NPE) and has a register file (RF) with 64 16-bit words that can be used for computation. This improves energy efficiency as the access energy cost is smaller than the SRAM memory. When in computation mode, all NPEs work in lock-step mode, executing the same instruction at any given cycle similar to a single instruction, multiple data (SIMD) operation.

As a neuromorphic platform, SENECA generates *ac./sp.* outputs through the NPEs when they meet certain conditions according to the workload. In this work, if the neuron state is bigger than the threshold, an *ac./sp.* would be generated. These *ac./sp.* are then processed by the event capture unit, which converts the input *ac./sp.* vector into address event representation (AER) format (Yousefzadeh et al., 2017). The event capture unit sends an interrupt to the RISC-V controller for further processing whenever a new *ac./sp.* is generated. In this work, generated activation(s)/spike(s) are encoded event(s) and then transmitted to another core through the NoC.

The RISC-V controller decides which operations should be executed on the NPEs depending on the workload scheduling. The loop controller coordinates the time-multiplexing of NPEs and the address generation for data memory access. It dispatches microcodes to the NPEs, enabling the processing of events. Each microcode is invoked to handle a specific type of event, such as neuron updates, threshold evaluations, or data conversions. For a more in-depth review of the SENECA architecture, please refer to Tang, Safa et al. (2023), Tang, Vadivel et al. (2023), Yousefzadeh et al. (2022) and Xu et al. (2024).

To optimize the processing of sparse data flows between layers of neurons, SENECA executes event-driven neural processing. There are different types of events, where each type triggers a specific set of computations, such as binary spikes produced by spiking neurons, non-zero activations generated by the ReLU/FATReLU activation function, and the synchronization signal representing the end of the time step

or data frame. When an *ac./sp.* is received, an event-integration task is executed. For ANN, the event-integration task multiplies the activation value with the corresponding weight vector and integrates the results into the neuron state vector. For SNN, the corresponding weight vector is integrated into the neuron state vector. It has been demonstrated in Tang, Vadivel et al. (2023) that integrating activation results in minimal energy overhead than integrating binary spikes. When a synchronization event is received, an event generation task is executed. The neurons that have not been processed by the event generation task yet will be processed. As to be introduced in Appendix D, for a *conv* layer, a neuron would be processed by the event generation task as soon as its state has been updated by the last *ac./sp.* in its receptive field. So a synchronization event will trigger the processing of the rest neurons in the lower part of the feature map. For ANN, the event generation task applies the activation function, *e.g.* FATReLU, to neuron states and generates non-zero activation events if there is neuron state(s) passing the thresholding. For SNN, the firing threshold is compared with the neuron membrane voltage to decide whether to generate a binary spike.

In general, event-driven processing for neural networks includes three phases:

- **Event Reception**: Unpack/decode the event and prepare for neural processing based on the information carried by the event and the recipient neurons.
- **Neural Processing**: Execute neurosynaptic computations and update neuron states.
- **Event Transmission**: Pack the generated spikes in one event packet and multi-cast it to the destination core(s).

During neural network computation, an event received from the NoC wakes up the RISC-V controller in a SENECA core and triggers the event reception phase. According to the decoded event, the event reception function determines the type of neural processing required and defines a set of executable tasks (which are represented by microcode to be executed at the NPEs). The loop controller receives the tasks and controls the time-multiplexed neural processing steps in the NPEs. The loop controller operates asynchronously with the event reception functions, allowing for accelerated and parallelized event processing. If a task execution involves event generation from neuron states, the event generator collects non-zero outputs from NPEs and packs them as AER events. These events then wake up the RISC-V controller and trigger the event transmission phase, which encodes and packages the AER events as compressed network event packets. Finally, the network event packet is sent to the destination core(s) through the NoC. Neural processing through the loop controller can work in parallel with the event reception/transmission processes since the loop controller can orchestrate the neural processing independently from the RISC-V controller.

Appendix D. Event-driven depth-first convolution on SENECA

Same as the previous section, this section was also introduced in Xu et al. (2024).

Fig. 9 shows the differences between the standard and event-driven convolution. The latter processes input *ac./sp.* one by one in their order of arrival and integrate them incrementally into the neuron states of the corresponding fanned-out postsynaptic neurons. However, this process requires maintaining high-dimensional neuron states of convolutional layers in memory, which is impractical for the limited size of the on-chip memory when the output tensor has a high dimension. To overcome this challenge, we propose the event-driven depth-first convolution.

Depth-first inference (Mei, Goetschalckx, Symons, & Verhelst, 2023; Waeijen, Sioutas, Peemen, Lindwer, & Corporaal, 2021) is a scheduling method in neural network inference that prioritizes the network's layer (depth) dimension by consuming *ac./sp.* right after their generation. In our event-driven depth-first inference, the input *ac./sp.* within a time step are assumed to be sorted in spatial order from the top-left corner of the (X, Y) plane to the bottom-right corner. Under this assumption, a neuron will receive all of its input *ac./sp.*s in a pre-defined order. Accordingly, its neuron state will be concluded earlier than those of spatially lower-ranked neurons (Fig. 10). As a result, it can fire (conduct the event generation process) immediately after its last neuron state updating without waiting for all the input *ac./sp.* of the layer are processed.

After firing an ANN neuron, its neuron state is no longer required to be maintained in the memory and thus the corresponding memory block can be released. For event-driven depth-first convolution of ANN, each layer only needs to buffer a small portion of neuron states that are incomplete/partially summed (the amount of required memory increases with the kernel size). For an SNN neuron, the situation is different due to the necessity of maintaining the neuron membrane potential for the next time step. Its membrane potential is reset to zero if the membrane potential is bigger than the firing threshold, otherwise, it is multiplied with the leak parameter to get ready for the next time step. The memory for the neuron states cannot be released like ANN, *i.e.* SNN needs to maintain all neuron states in the memory while ANN only needs to maintain several rows of neurons (for FireNet, three rows, given the 3×3 convolution kernels). This leads to the fact that, when using the same memory size, ANNs can accommodate a convolutional layer whose feature map has a much bigger dimension than SNN. However, this only applies to the situation when a convolutional layer has no recurrence. For the FireNet network architecture studied in this work, the ANN has two recurrent blocks whose hidden states must be stored in the memory for the following time steps. Since we use one SENECA core to accommodate a recurrent block, the maximum resolution is constrained by the memory size of one core, the same as an SNN layer. Therefore, the maximum resolution of ANN and SNN are the same.

Fig. 10 shows an example of the event-driven depth-first ANN convolutional layer with 3×3 kernels and 2×2 max pooling, more generic than the simple FireNet architecture (without pooling). It requires storing $(K + 1)$ lines of neuron states, equal to $X \times C \times (K + 1)$ neurons, where X is the spatial resolution (width), C is the number of channels, and K is the width of the kernel. In Fig. 10 where $K = 3$, neurons that are below the line $(X + 1)$ do not need to be stored because they have not received any activation yet. Similarly, neurons that are above the line $(X - 2)$ also do not need to be stored because they have already fired and do not expect to receive any more activation. Compared to storing all the neuron states $(X \times Y \times C)$ in the on-chip memory as required by SNNs, the memory requirement for ANN neuron states is significantly reduced. For FireNet studied in this work, the ANN convolution stores $56 \times 3 \times 32$ neuron states while the SNN convolution stores $56 \times 56 \times 32$ neuron states. This strategy enables the mapping of an ANN convolutional layer with a high spatial resolution to one SENECA core.

In event-driven depth-first convolution, the cycle of **event reception**, **neural processing**, and **event transmission** is executed as a *tail-recursion* for each 2D coordinate (pixel location). Fig. 10 illustrates the detailed procedure of our proposed event-driven depth-first convolution on a fused convolutional (kernel size 3×3, stride 1) and max-pooling (kernel size 2×2, stride (2) layer. We can divide this procedure into the following phases:

- When an event from the input location (x, y) has been received, all the neuron states above the $(y-1)th$ row or on the left of location $(x-1, y-1)$ will not be updated further because there will not be any future incoming event that is within the kernel window view (3×3 receptive field). Therefore, the event generation task will be triggered to generate the respective post-synaptic layer *ac./sp.* and then free up the memory storing the neuron states (for ANN) or perform the leak operation for neuron states (for SNN).

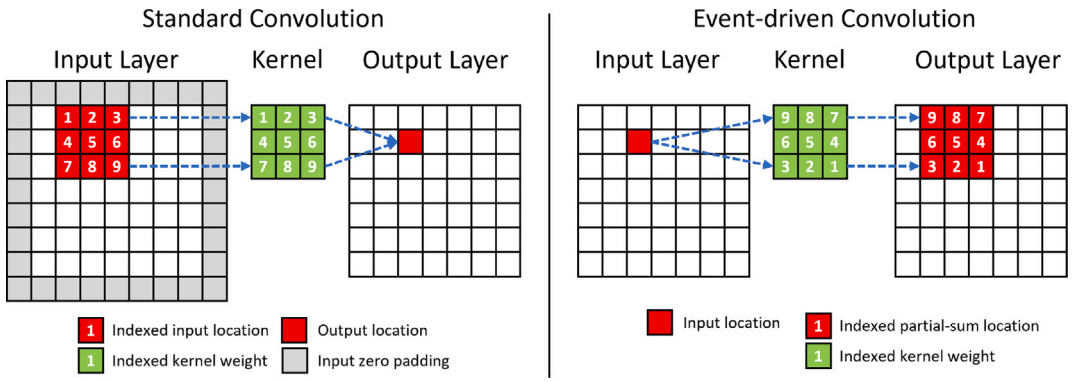


Fig. 9. Comparison between the standard and the event-driven convolution. The event-driven convolution requires to rearrange the sequence of kernel weights. The change in the spatial sequence for a 3 × 3 convolution kernel is shown in the figure. The channel dimension of the tensor is omitted for simplicity.

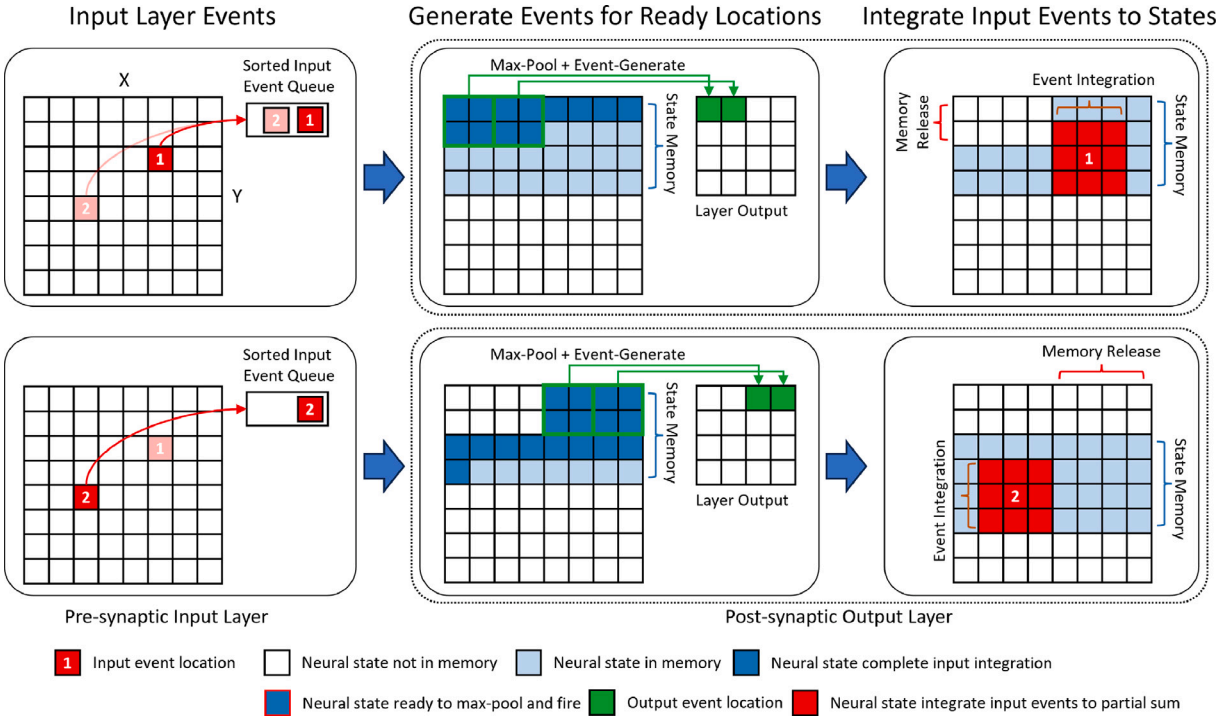


Fig. 10. An illustration of the event-driven depth-first convolution on SENECA. We show a fused layer combining 3 × 3 ANN convolution and 2 × 2 max-pooling. The layer processes input activations sequentially from the sorted input activation queue. Based on the pixel location of the new coming activation, event(s) are generated from a pixel location that has just been fully updated, and the corresponding memory spaces of the neurons at this pixel location are released. The channel dimension is omitted for simplicity.

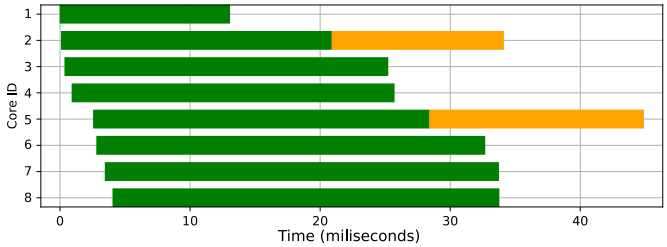


Fig. 11. Activated time of the 8 SENECA cores running the SNN FireNet processing the 56 × 56 testing frame with ~100% average neuron density. The color green indicates the processing time of the forward convolutional layer and orange corresponds to the recurrent convolutional layer.

- After firing the fully updated neurons, the input events at location (x, y) are processed. As a result, post-synaptic activity is generated at the same time as the input event trace is being processed. The event integration task integrates an input activation value to the neuron states within the 3 × 3 spatial locations around the input location (x, y).
- If the neuron states at a spatial location have been fully updated, for ANN, the event generation task applies the activation function (*e.g.*, ReLU/FATReLU) and 2 × 2 max-pooling function to the neuron states to generate non-zero activation events. For SNN, the neuron membrane potential is compared with the firing threshold to generate binary spike events. The event transmission function packs the *ac./sp.* from the same spatial neuron location (pixel location) into an event stream with shared header information of pixel coordinates and the number of *ac./sp.*. For ANN activations, each activation is encoded into an event, with 16 bits for the channel index and the rest 16 bits for its value (BFloat16). For SNN spikes, we use a bit to encode whether a channel has a spike. If there is a spike at channel c, then the c bit will be 1, otherwise 0. For FireNet, the number of channels is 32 so an event (32 bits) can encode all channels. For a network with more channels, more

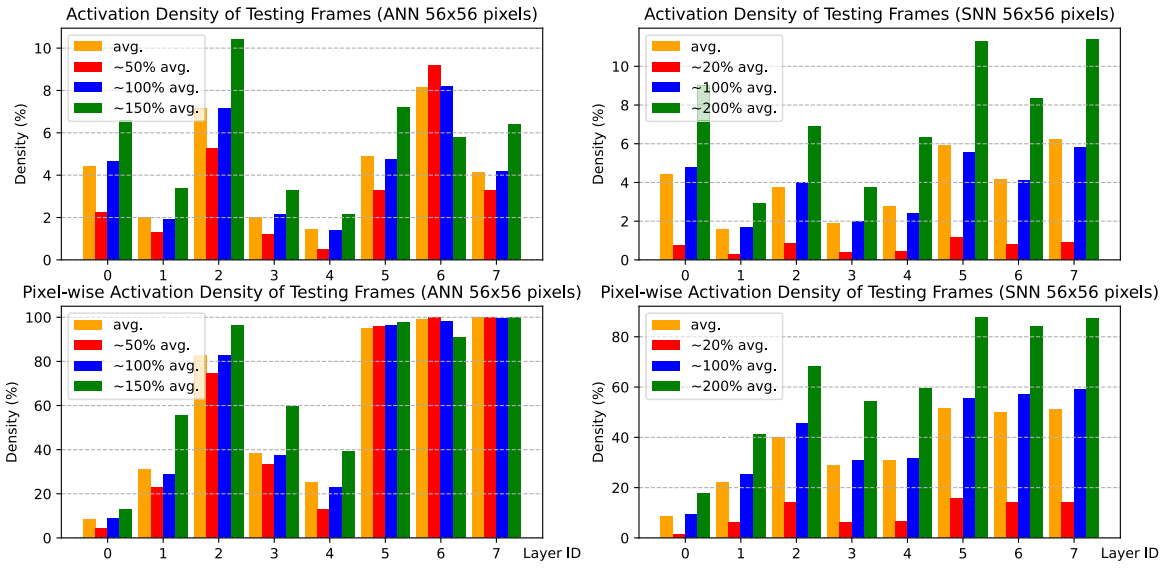


Fig. 12. Layer-wise density of the testing frames with resolution 56 × 56 pixels.

than one event is required. The event stream is then sent to the destination cores through the NoC.

Event-driven depth-first convolution can significantly reduce the inference latency when performing layer-to-layer event-driven dataflow processing in hardware. Traditional event-driven neuromorphic processing requires *barrier* synchronization at the end of the time step before event generation and communication. This introduces an additional latency per layer that equals the time required to integrate all the input events before a neuron can fire. The lock-step processing of event reception and neural processing in depth-first convolution enables multilayered parallelism in a pipelined fashion across layers without the need for explicit per-timestep barrier synchronization primitives. As shown in Fig. 11, the 8 SENECA cores running the SNN FireNet are highly parallelized.

A limitation of the proposed event-driven depth-first convolution is that it requires the input *ac./sp.* to be sorted and arrive in order. When using a conventional frame-based camera, this requirement is automatically satisfied. However, an additional process is needed to sort the input events of the first layer when dealing with the asynchronous measurements from an event-based camera. Nonetheless, the overhead is minimal if the input events are sparse. In this work, the overhead is not counted in the measurement of time cost because it is the same for ANN and SNN.

Despite the advantages of event-driven processing, there are significant overheads during **event reception** (unpacking/decoding each event and preparing the task) and **neural processing** (read/write neuron states per each event). The time and resources required for these steps can easily dominate the overall costs. For example, as shown in Tang, Safa et al. (2023), a single memory access for the data movement from SRAM to registers can be more than twice the cost of an arithmetic instruction.

As mentioned, processing each *ac./sp.* requires the following steps: (1) event decoding or projecting the spike address to several physical addresses of the weights and neuron states in data memory, (2) reading the relevant weights and neuron states from the data memory, (3) performing the neural calculation, and (4) writing the updated neuron states to the data memory. To reduce the processing cost per *ac./sp.*, *ac./sp.* in the same time step and updating the exact same set of neurons of the next layer are combined as a group (in this work, the group size is 4). Packing such *ac./sp.* that share the same destination neuron addresses significantly reduces the overhead of event decoding (step 1). Then, during the neural processing step, a neuron's state can be read once and updated multiple times by the grouped multiple *ac./sp.* before it is stored back into memory, considerably reducing memory accesses (steps 2 and 4).

Appendix E. More details of selected event frames with 56 × 56 pixels

As introduced in Section 6.3, we select three frames with 56 × 56 pixels for the ANN and the SNN, respectively. Here we show more information about the payer-wise *ac./sp.* density produced by the selected frames in Fig. 12. Comparing the ANN (two subplots on the left) and the SNN (two subplots on the right), the SNN's neuron density and pixel density are much better correlated. In contrast, the ANN's pixel density is not well correlated with neuron density, especially for the last three layers whose pixel density is almost 100%. Because of the parallelization of the layers/cores shown in Fig. 11, those layers can be the bottleneck of the whole network inference. Therefore, as shown in Fig. 7, the three ANN testing frames have similar time costs.

Appendix F. Open-sourced code

The code developed for this work is available at link.

Data availability

Data will be made available on request.

References

Cao, S., Ma, L., Xiao, W., Zhang, C., Liu, Y., Zhang, L., et al. (2019). Seernet: Predicting convolutional neural network feature-map sparsity through low-bit quantization. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (pp. 11216–11225).

Chaney, K., Panagopoulou, A., Lee, C., Roy, K., & Daniilidis, K. (2021). Self-supervised optical flow with spiking neural networks and event based cameras. In *2021 IEEE/RSJ international conference on intelligent robots and systems* (pp. 5892–5899). IEEE.

Chen, Y.-H., Yang, T.-J., Emer, J., & Sze, V. (2019). Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, *9*(2), 292–308.

Cuadrado, J., Rançon, U., Cottereau, B. R., Barranco, F., & Masquelier, T. (2023). Optical flow estimation from event-based cameras and spiking neural networks. *Frontiers in Neuroscience*, *17*, Article 1160034.

Dampfhoffer, M., Mesquida, T., Valentian, A., & Anghel, L. (2022). Are SNNs really more energy-efficient than ANNs? an in-depth hardware-aware study. *IEEE Transactions on Emerging Topics in Computational Intelligence*.

Davidson, S., & Furber, S. B. (2021). Comparison of artificial and spiking neural networks on digital hardware. *Frontiers in Neuroscience*, *15*, Article 651141.

Davies, M., Wild, A., Orchard, G., Sandamirskaya, Y., Guerra, G. A. F., Joshi, P., et al. (2021). Advancing neuromorphic computing with loihi: A survey of results and outlook. *Proceedings of the IEEE*, *109*(5), 911–934.

Delmerico, J., Cieslewski, T., Rebecq, H., Faessler, M., & Scaramuzza, D. (2019). Are we ready for autonomous drone racing? the UZH-FPV drone racing dataset. In *2019 international conference on robotics and automation* (pp. 6713–6719). IEEE.

Deng, L., Wu, Y., Hu, X., Liang, L., Ding, Y., Li, G., et al. (2020). Rethinking the performance comparison between SNNS and ANNS. *Neural Networks*, *121*, 294–307.

Furber, S., & Bogdan, P. (2020). *Spinnaker-a spiking neural network architecture*. Now publishers.

Gallego, G., Gehrig, M., & Scaramuzza, D. (2019). Focus is all you need: Loss functions for event-based vision. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (pp. 12280–12289).

Gehrig, M., Millhäusler, M., Gehrig, D., & Scaramuzza, D. (2021). E-raft: Dense optical flow from event cameras. In *2021 international conference on 3D vision* (pp. 197–206). IEEE.

Georgiadis, G. (2019). Accelerating convolutional neural networks via activation map compression. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (pp. 7085–7095).

Grimaldi, M., Ganji, D. C., Lazarevich, I., & Sah, S. (2023). Accelerating deep neural networks via semi-structured activation sparsity. In *Proceedings of the IEEE/CVF international conference on computer vision* (pp. 1179–1188).

Hagenaars, J., Paredes-Vallés, F., & De Croon, G. (2021). Self-supervised learning of event-based optical flow with spiking neural networks. *Advances in Neural Information Processing Systems*, *34*, 7167–7179.

Hoefler, T., Alistarh, D., Ben-Nun, T., Dryden, N., & Peste, A. (2021). Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. *Journal of Machine Learning Research*, *22*(241), 1–124.

Horowitz, M. (2014). 1.1 computing's energy problem (and what we can do about it). In *2014 IEEE international solid-state circuits conference digest of technical papers* (pp. 10–14). IEEE.

Hoyer, P. O. (2004). Non-negative matrix factorization with sparseness constraints. *Journal of Machine Learning Research*, *5*(9).

Kosta, A. K., & Roy, K. (2023). Adaptive-spikenet: event-based optical flow estimation using spiking neural networks with learnable neuronal dynamics. In *2023 IEEE international conference on robotics and automation* (pp. 6021–6027). IEEE.

Kurtz, M., Kopinsky, J., Gelashvili, R., Matveev, A., Carr, J., Goin, M., et al. (2020). Inducing and exploiting activation sparsity for fast inference on deep neural networks. In *International conference on machine learning* (pp. 5533–5543). PMLR.

Lee, H., Kim, C., Lee, S., Baek, E., & Kim, J. (2021). An accurate and fair evaluation methodology for SNN-based inferencing with full-stack hardware design space explorations. *Neurocomputing*, *455*, 125–138.

Lee, C., Kosta, A. K., Zhu, A. Z., Chaney, K., Daniilidis, K., & Roy, K. (2020). Spike-flownet: event-based optical flow estimation with energy-efficient hybrid neural networks. In *European conference on computer vision* (pp. 366–382). Springer.

Lemaire, E., Miramond, B., Bilavarn, S., Saoud, H., & Abderrahmane, N. (2022). Synaptic activity and hardware footprint of spiking neural networks in digital neuromorphic systems. *ACM Transactions on Embedded Computing Systems*, *21*(6), 1–26.

Li, Z., You, C., Bhojanapalli, S., Li, D., Rawat, A. S., Reddi, S. J., et al. (2023). The lazy neuron phenomenon: On emergence of activation sparsity in transformers. In *The eleventh international conference on learning representations*.

Mei, L., Goetschalckx, K., Symons, A., & Verhelst, M. (2023). DeFiNES: Enabling fast exploration of the depth-first scheduling space for DNN accelerators through analytical modeling. In *2023 IEEE international symposium on high-performance computer architecture* (pp. 570–583). IEEE.

Modha, D. S., Akopyan, F., Andreopoulos, A., Appuswamy, R., Arthur, J. V., Cassidy, A. S., et al. (2023). Neural inference at the frontier of energy, space, and time. *Science*, *382*(6668), 329–335.

Parashar, A., Rhu, M., Mukkara, A., Puglielli, A., Venkatesan, R., Khailany, B., et al. (2017). SCNN: An accelerator for compressed-sparse convolutional neural networks. *ACM SIGARCH Computer Architecture News*, *45*(2), 27–40.

Paredes-Vallés, F., Hagenaars, J., Dupeyroux, J., Stroobants, S., Xu, Y., & de Croon, G. (2023). Fully neuromorphic vision and control for autonomous drone flight. arXiv preprint arXiv:2303.08778.

Paredes-Vallés, F., Scheper, K. Y. W., De Wagter, C., & de Croon, G. C. H. E. (2023). Taming contrast maximization for learning sequential, low-latency, event-based optical flow. In *Proceedings of the IEEE/CVF international conference on computer vision* (pp. 9695–9705).

Ponghiran, W., Liyanagedera, C. M., & Roy, K. (2023). Event-based temporally dense optical flow estimation with sequential learning. In *Proceedings of the IEEE/CVF international conference on computer vision* (pp. 9827–9836).

Reddi, V. J., Cheng, C., Kanter, D., Mattson, P., Schmuelling, G., Wu, C.-J., et al. (2020). Mlperf inference benchmark. In *2020 ACM/IEEE 47th annual international symposium on computer architecture* (pp. 446–459). IEEE.

Rhodes, O., Bogdan, P. A., Brenninkmeijer, C., Davidson, S., Fellows, D., Gait, A., et al. (2018). SPyNNaker: a software package for running PyNN simulations on SpiNNaker. *Frontiers in Neuroscience*, *12*, 816.

Runwal, B., Pedapati, T., & Chen, P.-Y. (2023). Parameter efficient finetuning for reducing activation density in transformers. In *Annual conference on neural information processing systems*.

Schnider, Y., Woźniak, S., Gehrig, M., Lecomte, J., Von Arnim, A., Benini, L., et al. (2023). Neuromorphic optical flow and real-time implementation with event cameras. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (pp. 4128–4137).

Sengupta, A., Ye, Y., Wang, R., Liu, C., & Roy, K. (2019). Going deeper in spiking neural networks: VGG and residual architectures. *Frontiers in Neuroscience*, *13*, 95.

Shiba, S., Aoki, Y., & Gallego, G. (2022). Secrets of event-based optical flow. In *European conference on computer vision* (pp. 628–645). Springer.

Tang, G., Safa, A., Shidqi, K., Detterer, P., Traferro, S., Konijnenburg, M., et al. (2023). Open the box of digital neuromorphic processor: Towards effective algorithm-hardware co-design. In *2023 IEEE international symposium on circuits and systems* (pp. 1–5). IEEE.

Tang, G., Vadivel, K., Xu, Y., Bilgic, R., Shidqi, K., Detterer, P., et al. (2023). SENECA: building a fully digital neuromorphic processor, design trade-offs and challenges. *Frontiers in Neuroscience*, *17*.

Waeijen, L., Sioutas, S., Peemen, M., Lindwer, M., & Corporaal, H. (2021). ConvFusion: A model for layer fusion in convolutional neural networks. *IEEE Access*, *9*, 168245–168267.

Wu, Y., Deng, L., Li, G., Zhu, J., Xie, Y., & Shi, L. (2019). Direct training for spiking neural networks: Faster, larger, better. *Vol. 33*, In *Proceedings of the AAAI conference on artificial intelligence* (pp. 1311–1318).

Xu, Y., Shidqi, K., van Schaik, G.-J., Bilgic, R., Dobrita, A., Wang, S., et al. (2024). Optimizing event-based neural networks on digital neuromorphic architecture: a comprehensive design space exploration. *Frontiers in Neuroscience*, *18*, Article 1335422.

Yang, Q., Mao, J., Wang, Z., & Hai, H. L. (2021). Dynamic regularization on activation sparsity for neural network efficiency improvement. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, *17*(4), 1–16.

Yik, J., Berghe, K. V. d., Blanken, D. d., Bouhadjar, Y., Fabre, M., Hueber, P., et al. (2023). NeuroBench: A framework for benchmarking neuromorphic computing algorithms and systems. arXiv preprint arXiv:2304.04640.

Yousefzadeh, A., Jabłoński, M., Iakymchuk, T., Linares-Barranco, A., Rosado, A., Plana, L. A., et al. (2017). On multiple AER handshaking channels over high-speed bit-serial bidirectional LVDS links with flow-control and clock-correction on commercial FPGAs for scalable neuromorphic systems. *IEEE Transactions on Biomedical Circuits and Systems*, *11*(5), 1133–1147.

Yousefzadeh, A., Van Schaik, G.-J., Tahghighi, M., Detterer, P., Traferro, S., Hijdra, M., et al. (2022). SENeCA: Scalable energy-efficient neuromorphic computer architecture. In *2022 IEEE 4th international conference on artificial intelligence circuits and systems* (pp. 371–374). IEEE.

Zhu, A. Z., Thakur, D., Özaslan, T., Pfrommer, B., Kumar, V., & Daniilidis, K. (2018). The multivehicle stereo event camera dataset: An event camera dataset for 3D perception. *IEEE Robotics and Automation Letters*, *3*(3), 2032–2039.

Zhu, A. Z., & Yuan, L. (2018). EV-FlowNet: Self-supervised optical flow estimation for event-based cameras. In *Robotics: science and systems*.

Zhu, A. Z., Yuan, L., Chaney, K., & Daniilidis, K. (2019). Unsupervised event-based learning of optical flow, depth, and egomotion. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (pp. 989–997).