SCIENCE CHINA
Information Sciences

• Supplementary File •

EE-Extractor: A Near-sensor Real-time Effective Event Extractor for Dynamic Vision Sensor

Feiqiang Li[1,†], Yujie Huang[1,†*], Mingyu Wang[1*], Wenhong Li[1*], Minge Jing[1*] & Xiaoyang Zeng[1*]

[1]*State Key Laboratory of Integrated Chips and Systems, Fudan University, Shanghai 200000, China.*
[†]*Feiqiang Li and Yujie Huang have the same contribution to this work.*

Appendix A Proposed algorithm

Appendix A.1 Bit Queue spatiotemporal filter

With the development of Dynamic Vision Sensor (DVS), the request of events, the readout of response, and the reset of the pixel array are mostly based on rows [1, 2]. As shown in the left half of Figure A1, the pixel array outputs the events one row at a time as well as the row number if there are events in that row. ON events and OFF events are N bits each, where 0 represents no event, and 1 indicates that an event has occurred at that location. After the $P2S$ (parallel-to-serial) module, one row of events is converted into discrete events. Currently, for high-resolution DVS, all the spatiotemporal filters are implemented after the interface circuit and based on discrete events. They ignore the row position relationship between events that have been arbitrated out, which leads to a waste of storage resources.

Therefore, leveraging the row-wise arbitration characteristics of the DVS pixel array output, a Bit Queue is designed for the proposed filter. Rather than storing event coordinates (x, y), Bit Queue directly stores one-row events from the DVS array, where 1 indicates the presence of the event. As illustrated in Figure A2(a), the Bit Queue maintains a sliding window of the most recent R rows of events, with the corresponding row numbers stored in Row Num. For the proposed filter, spatiotemporal correlation is constrained within the Bit Queue. This implementation transforms the time threshold dT into the time interval between the current row and the initial row in the queue ($dT = t_R - t_1$). Therefore, an event is validated as real if adjacent events are found within this queue. Storing eight rows of data requires only $8 \times 512 + 8 \times 9$ bits = 4.072 kb, which represents less than 25% of the memory needed by the Queue-based filter [3].

As shown in Figure A2(a), for denoising, the filter extracts an adjacent event matrix E centered at the current event coordinate (x_i, y_i) from the Bit Queue. The matrix E covers a spatial neighborhood defined by $x \in [x_i - D, x_i + D]$ and $y \in [y_i - D, y_i + D]$, where D is the denoising threshold of the filter. If there is "1" in the adjacent events matrix E, the current event is considered a real event. Compared to [3], this approach significantly reduces computational complexity by using localized event comparisons instead of global ones while maintaining equivalent denoising performance. Binary representation and constrained search space enable efficient hardware implementation with minimal memory requirements.

Appendix A.2 Boundary-box-based clustering

To overcome the limitations of existing event-based clustering algorithms, boundary-box-based clustering is proposed in this paper. Firstly, a boundary-box-based rectangle representation of the category is proposed. It avoids the error caused by centroid-and-radii-based square representation [4], and has lower computational complexity than Queue-based clustering [3]. Then, to reduce hardware overhead, a category coverage mechanism is presented. To ensure the effectiveness of clustering, a clustering ending mechanism is proposed. In the following, the clustering will be introduced from the representation of categories, the process of clustering, the category coverage mechanism, and the clustering ending mechanism.

Appendix A.2.1 *Representation of categories*

As shown in Figure A2(b), the category is represented by the boundary box and its event count, which also serves as the fundamental unit for event-by-event clustering. An event is assigned to a category when its distance to the boundary box falls below the clustering threshold G. Following this assignment, the system dynamically updates both the boundary box and the event count for the category. In principle, the rectangular representation of the category avoids the error caused by the centroid-and-radii-based square representation. Moreover, by requiring only boundary comparisons for category assignment, the method achieves lower computational complexity than Queue-based clustering, thereby optimizing the trade-off between clustering accuracy and processing efficiency. The complete clustering system comprises multiple such clustering units, and their collaboration can efficiently complete real-time clustering.

* Corresponding author (email: mejing@fudan.edu.cn, yujiehuang@fudan.edu.cn, mywang@fudan.edu.cn, xyzeng@fudan.edu.cn, wenhongli@fudan.edu.cn)

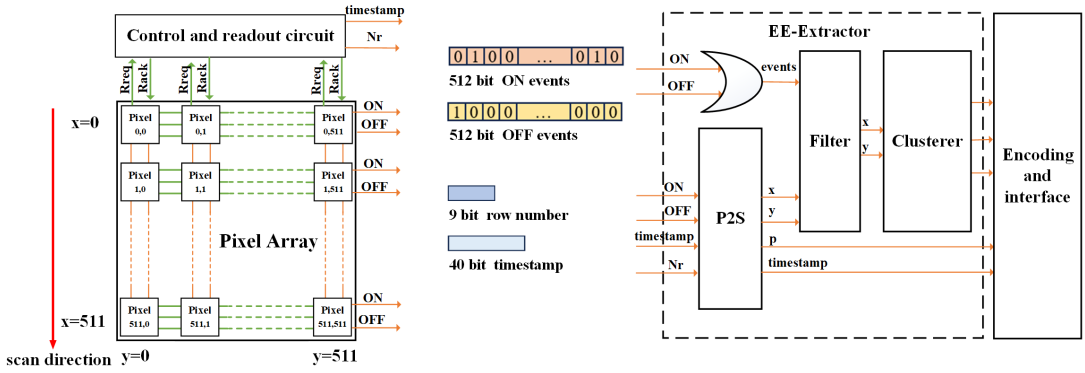


Figure A1 The figure shows the DVS with a resolution of 512 × 512. The proposed EE-Extractor is implemented near the DVS pixel array, including a Bit Queue spatiotemporal filter and a boundary-box-based clusterer. For the DVS, events are read from top to bottom in a row-scanning mode, and the row without events is not read. Events in a row are 512-bit ON events (0 represents no event and 1 indicates that an event has occurred at this location), 512-bit OFF events, 40-bit timestamp shared by the current row, and 9-bit row number Nr of the current row.

Appendix A.2.2 *Process of clustering*

The proposed clustering architecture is illustrated in Figure A2(b), where (x, y) represents a denoised event. Category information memory stores information on clustered events, including the category's boundaries ($xmax$, $xmin$, $ymax$, and $ymin$) and the event count $lcount$. Clustered events are stored in Event memory. The specific steps of the boundary-box-based clustering are as follows:

1. Initialize Category information memory for all n available categories.

2. For each incoming event (x, y), assign it to category i if it satisfies (A1).

$$\begin{aligned} e(x,y) \in i \Leftrightarrow & (x < xmax[i] + G) \wedge (x > xmin[i] - G) \wedge (y < ymax[i] + G) \\ & \wedge (y > ymin[i] - G) \wedge (lcount[i] > 0), \quad i \; in \; \{1, ..., n\} \end{aligned} \tag{A1}$$

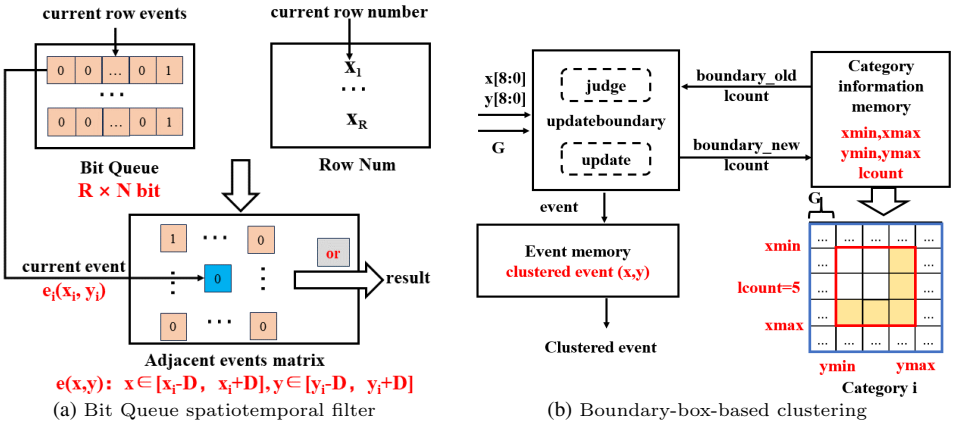
3. Following category assignment, the system executes boundary updates according to Algorithm A1. The update procedure handles three cases. If the event matches no existing category and idle categories are available, it initializes a new category. An idle category is defined as either containing zero events or that can be covered (see the Category coverage mechanism). If the event falls within one or more existing categories, the system merges these categories and updates the categories' boundaries if needed. In others, the event is discarded as noise.

Algorithm A1 Flow of the boundary-box-based clustering

Input: Event $e(x,y)$
Result: Update category information: $xmin$, $ymin$, $xmax$, $ymax$, $lcount$

if $e \notin category\ i$, $s.t. \forall\ i$ in $\{1, ..., n\}$ **then**
  if the idle categories i_1,...,i_z exist **then**
    $cmin = min\{i_1,...,i_z\}$;
    Initialize category $cmin$;
  else
    event e is discarded;
  end if
else
  $e \in$ category k_1,...,k_x, $cfirst = min\{k_1,...,k_x\}$;
  Update boundary and $lcount$ of category $cfirst$
  for k in $\{k_1,...,k_x\} \wedge k \neq cfirst$ **do**
    Clear category k;
  end for
end if

4. When the ending condition of clustering is reached (see the Clustering ending mechanism), output the category whose event number exceeds the threshold LN. Then, events generated from different objects are distinguished, and noise clusters with fewer than LN are further removed.



(a) Bit Queue spatiotemporal filter (b) Boundary-box-based clustering

Figure A2 (a) is the block diagram of the Bit Queue spatiotemporal filter. The Bit Queue stores the past R rows of events, with each row represented by N bits. Row Num stores the row numbers. D is the distance threshold of the filter. For the current event (x_i, y_i), the adjacent events matrix E represents the occurrence of events in the vicinity of (x_i, y_i), which is taken from the Bit Queue. If there is "1" in the adjacent events matrix, the current event is considered a real event. (b) is the block diagram of the boundary-box-based clustering. G is the threshold of clustering. Category information memory saves information on clustered events, including the category's boundaries and the number of events $lcount$. Clustered events are stored in Event memory. The red box in the bottom right corner of the figure represents category i, and the blue box represents the extension box of the category.

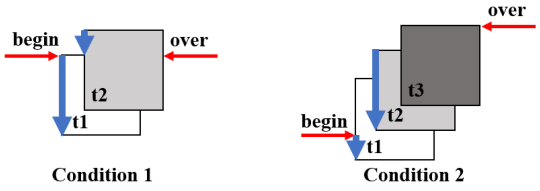


Figure A3 Two conditions for clustering ending mechanism, taking the upward motion of an object as an example, where $t3 > t2 > t1$. The trajectory scanned by DVS in rows is shown by the blue arrow.

Appendix A.2.3 *Category coverage mechanism*

During clustering, several categories would be generated, particularly in high-noise scenarios, making it challenging to determine an optimal fixed number of categories. Setting the category count n too large leads to two drawbacks: increased memory overhead for category information, and higher computational complexity during clustering. When setting a small n, many subsequent events cannot be attributed to the existing categories, resulting in the loss of real events. To address this, a novel category coverage mechanism is proposed, which enables effective clustering with a relatively small n.

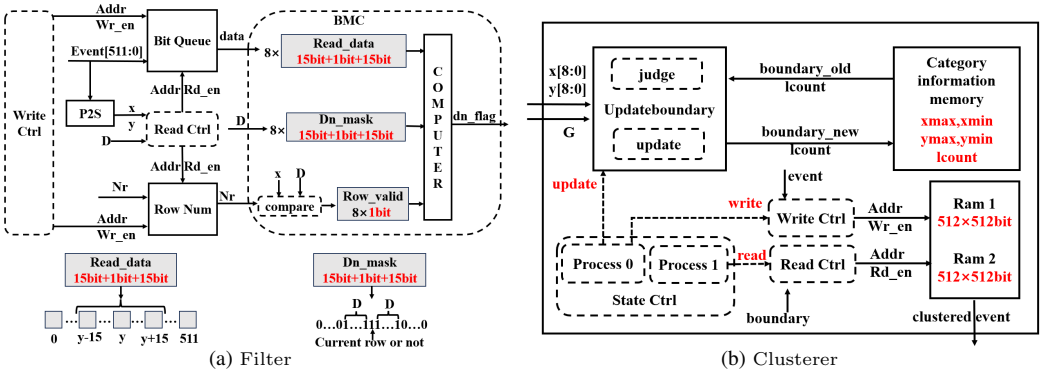
When an event does not belong to any category and all categories have events, it is necessary to determine which categories can be covered. The easiest way is that the category with few events can be covered. However, the categories are gradually clustered. If only considering the event's number of categories, it is possible to mistakenly cover the category whose number of events is gradually increasing. Therefore, based on the output characteristics of the DVS pixel array, we propose that categories, whose number of events has not increased over a period of time and are less than LN, can be covered, where the number of events less than LN is the judgment of noise clusters after clustering. Besides, since DVS outputs in row arbitration mode, the time in the coverage condition can be equivalent to the number of rows scanned, as shown in (A2). Due to the sequential arbitration of DVS, when DVS has arbitrated to a certain distance G from the boundary $xmax$ of the category, it indicates that the number of categories will no longer increase during the current scanning cycle (from top to bottom). If there are fewer than LN events in the category at this time, the category is considered a noise cluster and can be covered.

$$\begin{cases} x - xmax[i] \geqslant G, & \text{if } \ xmax[i] \leqslant x \\ 512 + x - xmax[i] \geqslant G, & \text{otherwise.} \end{cases} \tag{A2}$$

Appendix A.2.4 *Clustering ending mechanism*

A reasonable clustering ending mechanism is important for clustering. The premature end of clustering may lead to incomplete clustering of objects, which is not conducive to subsequent recognition or tracking. If clustering ends too late, it may lead to ghosting or clustering different objects together. Ending clustering based on the number of clusters cannot adapt to various scenarios. In some scenarios, objects generate more events, while in others, they generate fewer. In this case, different number thresholds are required. Therefore, for clustering, based on the output characteristics of the DVS pixel array, a novel clustering ending mechanism is proposed.

Clustering requires extracting the complete shape of the objects. Due to the high time resolution of DVS, relatively complete object-generated events can be captured after a complete top-to-bottom row scan. Based on this, two termination



(a) Filter (b) Clusterer

Figure B1 (a) is the block diagram of the filter. Eight register groups are needed in *Bit Queue*, and eight registers are used in *Row Num* to store the row number of events in the *Bit Queue*. According to the *BMC* module, whether there exist events adjacent to the current event can be judged. (b) is the block diagram of the clusterer. *Category information memory* stores information on the boundaries and number of events for categories. *Updateboundary* is responsible for determining the category of events and updating the category's boundaries.

conditions were established for the clustering, as shown in Figure A3. Condition one is that when a new round of scanning has started, clustering ends at the latest event's row number being greater than or equal to the initial event's. Condition 2 is a supplement to Condition 1. When the latest event's row number cannot be greater than the initial event's, clustering ends at the beginning of the next round of scanning. Moreover, when events are sparse, to accumulate more events to make objects more complete, this paper sets a minimum time threshold Tn for clustering. A maximum time threshold Tx has also been set to force the end of clustering, which prevents the clustering from failing to complete normally. In other words, the clustering time should be greater than Tn and less than Tx.

Appendix B Hardware implementation

Figure A1 shows the hardware implementation of the EE-Extractor. This paper takes the DVS with a resolution of 512×512 as an example to introduce the proposed EE-Extractor. The $P2S$ module converts the data from the DVS pixel array into individual events. The *Filter* module is the implementation of the Bit Queue spatiotemporal filter. The *Clusterer* module is the implementation of the boundary-box-based clustering. Finally, output processed events. The details of the *Filter* and *Clusterer* are as follows.

Appendix B.1 Implementation of the Filter

For the *Filter*, as shown in Figure B1(a), it mainly consists of the *Bit Queue*, *Row Num*, *Write Ctrl*, *Read Ctrl*, and *BMC* module. The *Bit Queue* is the memory of the filter, which stores events generated in the past R rows, and *Row Num* stores the row number of the events. *Write Ctrl* controls the write of the *Bit Queue* and *Row Num*. *Read Ctrl* is responsible for extracting adjacent events of the current event from *Bit Queue*. According to *BMC*, it can be determined whether the current event is real. *Write Ctrl* and *Row Num* are easy to understand and will not be elaborated here. The details of the *Bit Queue*, *Read Ctrl*, and *BMC* modules are as follows.

Appendix B.1.1 *Bit Queue*

For the proposed Bit Queue spatiotemporal filter, whether the current event is a real event depends on whether there are adjacent events in the queue. To ensure that the events in the queue have a temporal correlation, the length of the queue should not be too large. Therefore, taking into account the performance of the algorithm and hardware resources, the length of the queue is set to eight. To reduce the delay of denoising, the read delay of data in the queue should be as low as possible. Therefore, eight register groups are used in *Bit Queue*, ensuring that adjacent events in eight rows can be read out within one cycle. Each register group stores 512 bits of events, and eight register groups are written in turns.

Appendix B.1.2 *Read Ctrl*

Read Ctrl is responsible for processing the read data and extracting adjacent events of the current event from it. Adjacent events in eight rows are read out within one cycle. As shown in Figure B1(a), according to y, through *Read Ctrl*, eight *Read_data*s are taken from the *Bit Queue*. The *Read_data* is the data near the y position of the current event. The denoising threshold D of the filter is 4 bits, thus the maximum supported setting for D is 15. which requires 31-bit data *Read_data* (i.e. [y-15,y+15]). For the proposed filter, the denoising threshold can be adjusted within the range of 1 to 15. Then, according to x, through *Read Ctrl*, eight Nrs are taken from the *Row Num*, which will be used to select adjacent rows in *BMC*.

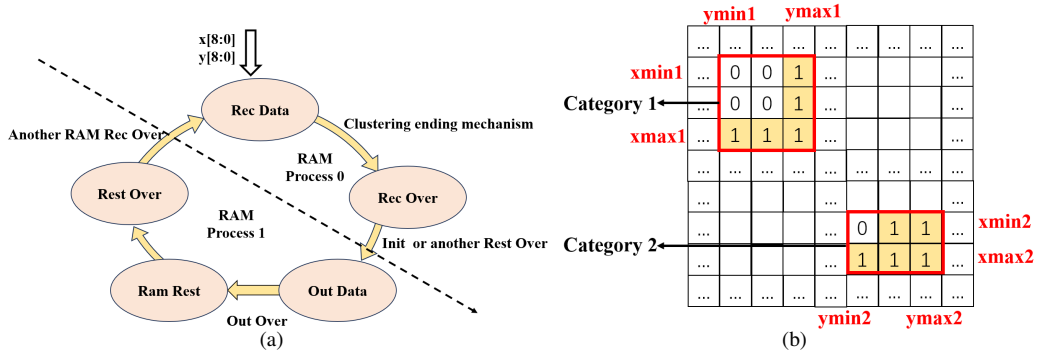


Figure B2 (a) is jumps in the state when clustering works. The upper right part of the state machine belongs to Process 0, and the lower left part belongs to Process 1. The two rams work alternately in two processes. (b) is the schematic diagram of *Ram* storing events. Categories 1 and 2 are the two categories after clustering. $xmax$, $xmin$, $ymax$, $ymin$ are the boundaries of the categories. "1" stored in *Ram* indicates that there is an event at that location.

Appendix B.1.3 *Bit Mask Computer*

For the designed Bit Queue, events are stored in the form of "bit", which provides convenience for simplifying operations. Therefore, the BMC is designed for the filter.

Firstly, prepare the data, as follows:

• Through the *Read Ctrl*, eight 31-bit data *Read_data*s are gotten (i.e. [y-15,y+15]), which are the data near y position of the current event.

• According to the threshold D, eight 31-bit *Dn_mask*s are gotten, which are used to select adjacent data within the D range. *Dn_mask*s and *Read_data*s correspond one-to-one. For example, if D is five, since the current event cannot prove itself, the *Dn_mask* for the current row is "0...0111110111110...0", and the *Dn_mask* for other rows is "0...0111111111110...0".

• According to the eight Nrs from $Row\ Num$, the threshold D and the position x, eight *Row_valid*s can be obtained. Let the *Row_valid* of the row that satisfies $Nr \in [x - D, x + D]$ equal to 1.

Then, according to (B1), it can be determined whether the current event is a real event, which can be completed with a small amount of AND or OR operations.

$$\begin{aligned} result[r] = (|(Read\_data[r]\&Dn\_mask[r]))\&(Row\_valid[r]), \quad r\ in\ \{1, ..., 8\} \\ dn\_flag = |result \end{aligned} \tag{B1}$$

Appendix B.2 Implementation of the clustering

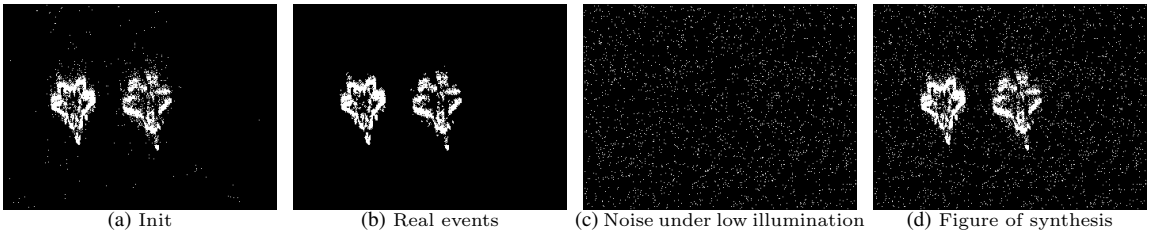
For the *Clusterer*, as shown in Figure B1(b), it mainly consists of the *Updateboundary*, *Category information memory*, *Ram*s, *Write Ctrl*, *Read Ctrl*, and *State Ctrl*. The module *Category information memory* stores information on the boundaries and number of events for n categories. Considering hardware complexity and algorithm effectiveness, the number of categories is set to eight. The module *Updateboundary* determines the categories of an event, and is responsible for updating categories' boundaries according to Algorithm A1. The clustered events are stored in *Ram*s, which are controlled by *Write Ctrl* and *Read Ctrl*. The module *State Ctrl* controls the state of the entire clusterer. The judgment and update of categories in *Updateboundary* have been introduced in Appendix A.2, and will not be elaborated here. In the following, the clusterer will be introduced from *State Ctrl* and a read-out way of clustered events.

Appendix B.2.1 *State Ctrl*

Only when clustering is completed, can the events be output according to the clustering results. The clustering needs to accumulate a certain number of events, and it also takes time to output the clustered events. Since the events stream is constantly generated, if the output of clustered events prevents it from receiving new events for clustering, it will result in event dropout and low throughput. Therefore, from the perspective of the entire system, this paper divides clustering into two processes, executing these two tasks in parallel. As shown in Figure B1(b), in Process 0, the system clusters the current event and updates the boundary box in real time. And in Process 1, the clustered events are read sequentially according to the boundary.

The clusterer is divided into two processes, which means two *Ram*s are needed to store the events after clustering, working alternately in Process 0 and Process 1, as shown in Figure B1(b). *Ram* 1 and *Ram* 2 form a ping-pong structure, and they work in parallel. When *Ram* 1 is in Process 0, storing the clustered events, *Ram* 2 works in Process 1, sending out the events sequentially. The state jumps of the clusterer are shown in Figure B2(a). Here is the specific implementation:

• For Process 0, module *Updateboundary* determines the category of the current event and updates the boundaries based on the location of the event and the boundaries of the existing categories. If the event is clustered, its location information is stored in one *Ram*. When the clustering end condition is met, Process 0 ends, and the state jumps from *Rec Data* to *Rec Over*.



(a) Init (b) Real events (c) Noise under low illumination (d) Figure of synthesis

Figure C1 (a) is the $5ms$ accumulation of events generated by the free fall of two airplane models under sufficient light conditions. (b) is the figure with only real events. (c) is the $5ms$ accumulation of noise under low light conditions. (d) is the synthetic picture of (b) and (c).

Table C1 The performance of filters under two illumination conditions.

illumination	Sufficient lighting				Low lighting			
performance	*Acc*	*Pre*	*Rec*	$F1_d$	*Acc*	*Pre*	*Rec*	$F1_d$
Liu's [7]	0.5724	0.9940	0.5303	0.6916	0.6771	0.9534	0.4958	0.6523
O(N) [8]	0.6956	0.9909	0.6695	0.7991	0.7728	0.9502	0.6629	0.7809
O(N^2) [9]	0.8303	0.9885	0.8219	0.8975	0.8415	0.9376	0.7934	0.8595
Queue [3]	0.8266	0.9889	0.8174	0.8950	0.7724	0.9543	0.6590	0.7796
our filter	0.8859	0.9716	0.9001	0.9345	0.8806	0.8971	0.9088	0.9029
filter+clusterer	0.8861	0.9737	0.8983	0.9345	0.8904	0.9483	0.8679	0.9063

Red indicates the highest, and blue is the second highest.

• For Process 1, the system reads out data from the *Ram*, which stores clustered events. According to the boundary and the event's number of the category, the category whose number of events is greater than LN outputs its events sequentially. Finally, after processing, the system clears *Ram*.

Appendix B.2.2 *Read-out ways of clustered events*

Generally, DVS cameras output events in the order they occur (or in an arbitrated order), which facilitates flexible post-processing of events. Besides, the proposed clusterer can output events in category order based on the results of clustering. Outputting events in category order can filter out events of interest by location or by number of events, and on the other hand, the separated categories can be fed directly into subsequent recognition or tracking algorithms, which can reduce the complexity of post-processing.

Binary frame is a commonly used encoding method for events before post-processing in DVS [5]. Therefore, 512×512 bit *Ram* is used to store the events after clustering, in which the presence or absence of an event is denoted by 1 and 0. Under the control of $Write\ Ctrl$, x of the clustered event is the address written to *Ram*, and y is the position in the word that needs to be written as 1. As shown in Figure B2(b), after the clustering is completed, 512×512 bits *Ram* stores the binary feature map. Then, based on the boundaries of each category, the events of each category can be read out separately in coordinate or in "bit" form. When outputting by coordinates, if the word read from *Ram* includes 1, then output its address (i.e. x) and position equal to 1 (i.e. y). When outputting events in "bit" form, simply read data out from the *Ram*. For example, for category 1 in Figure B2(b), output the data "001" in the first row, data "001" in the second row, and so on.

Appendix C Results of the experiment

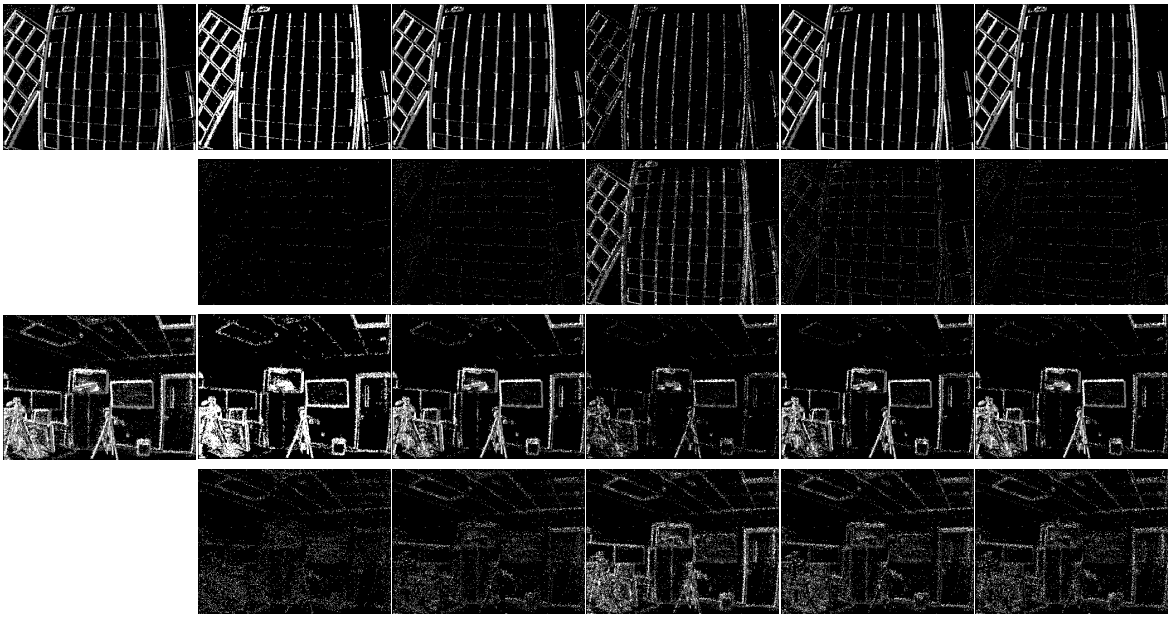
For the evaluation of the proposed EE-extractor, the experiment is divided into three parts. The first is the results of the proposed filter, including quantitative experiments and comparison with other filters on public datasets. The second is the results of the proposed clusterer, including its performance on denoising and comparison with other clusterers on ROI extraction for DVS. The third is the result of hardware implementation. The details of the experiment are as follows.

Appendix C.1 Results of filter comparison

Appendix C.1.1 *Quantitative experiment*

Since the existing datasets of DVS do not have accurate labels to calibrate the real events and noise, in this paper, synthetic datasets are collected to evaluate the algorithms quantitatively. In addition, due to the difficulty in obtaining event data before arbitration for commercial DVS, for the sake of comparison, the datasets need to be further processed based on the characteristics of DVS row scanning output. Firstly, the event stream captured by DAVIS346 [6] (resolution: 346×260) needs to be extended to a resolution of 512×512. Then, events on the same row within a certain period share a timestamp and are output by row scanning.

Under sufficient light conditions, DAVIS346 was used to photograph two airplane models. The DVS was stationary, and the airplane models fell freely in front of the DVS. Figure C1(a) is a figure accumulated from $5ms$ events, in which little noise is generated. Then, the method proposed in [3] was used to label the events (i.e. when there were more than two



(a) Init (b) Ours (c) O(N^2) [9] (d) O(N) [8] (e) Liu [7] (f) Queue [3]

Figure C2 "Init" are original images, including two scenes, and 50K original events are accumulated in them. The first rows of images of each scene are the events after being filtered. The second-row images of each scene are the events that are filtered out.

consecutive adjacent events in $5ms$, they were considered as real events). Under sufficient light conditions, little noise is generated by DVS. Therefore, this method of labeling real events and noise is effective. Using the above method, real events in Figure C1(a) can be obtained, as shown in Figure C1(b).

Under the low light condition, the static background is photographed with DAVIS346. Figure C1(c) is the $5ms$ accumulation of events. Because there was no change in light intensity, these events were considered as noise. Then, we combined the real events in Figure C1(b) with the noise in Figure C1(c). Figure C1(d) is the synthesized image.

Four parameters were used to evaluate the performance of the filters: Accuracy (Acc), Precision (Pre), Recall (Rec), and F1-score ($F1_d$). Acc reflects the accuracy of the filter. Pre reflects the proportion of real events in denoised events. Rec reflects the retention ratio of real events after passing through the filter. $F1_d$ is used to evaluate the overall performance of the filter. The larger the four parameters, the better the performance of the filter. The parameters are as follows:

$$Acc = \frac{S_{filtered} + N_{removed}}{A_{original}} \tag{C1}$$

$$Pre = \frac{S_{filtered}}{A_{filtered}} \tag{C2}$$

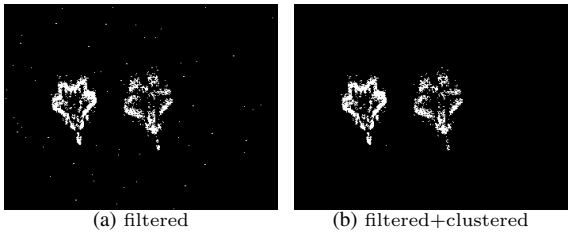
$$Rec = \frac{S_{filtered}}{S_{original}} \tag{C3}$$

$$F1_d = \frac{2 \times Rec \times Pre}{Rec + Pre} \tag{C4}$$

Here, $S_{filtered}$ is the number of real events passing the filter. $N_{removed}$ is the number of noise removed by the filter. $A_{original}$ is the number of all events in the original dataset. $A_{filtered}$ is the number of events that pass through the filter. $S_{original}$ is the number of real events in the original dataset.

The $O(N^2)$-Space spatiotemporal filter [9], $O(N)$-Space spatiotemporal filter [8], Liu's filter [7], Queue-based filter [3], and the proposed filter were compared in the experiment. For the proposed filter, set $D = 2$. To increase the persuasiveness of the experiment, the experiments used optimal experimental settings from previous works [3,7,8]. For $O(N^2)$, $O(N)$, and Liu's filter, set the time threshold to $1ms$, and for Liu's filter, set $S = 2$. According to statistics, for the dataset under sufficient illumination, the time of 968 events was about $1ms$. Therefore, the experiment set $M = 968$ for the Queue-based filter.

According to the experiment, the $O(N^2)$ filter, which requires the most memory, has good performance under two illuminations, as shown in Table C1. Its $F1_d$ is second only to our work. For the $O(N)$ filter and Liu's filter, due to their memory structure, they do not utilize complete spatiotemporal correlation and remove many real events, resulting in lower Rec and $F1_d$. For the Queue-based filter, it performs well under sufficient illumination. However, when light is insufficient, due to increased noise, it retains fewer real events, resulting in a significant decrease in Rec and $F1_d$. Compared to the above filters, the proposed filter remains the most real event, and it has the best performance under two lighting conditions. $F1_d$ of the proposed filter has been improved by more than 4.1% compared to the state-of-the-art method $O(N^2)$ while



(a) filtered (b) filtered+clustered

Figure C3 Under the low light condition, after being denoised by the proposed filter, the output is shown in Figure (a), in which noise cannot be completely removed. After passing through a filter and then clustering, the noise is almost completely removed, as shown in Figure (b).

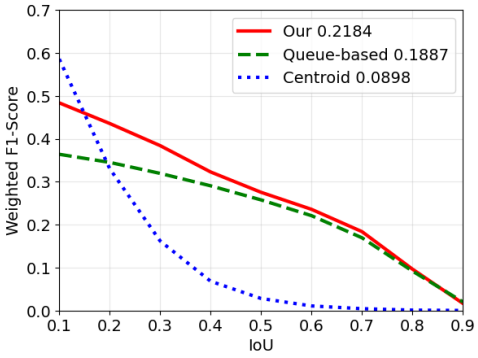


Figure C4 The figure shows the variation of the weighted F1-score with IoU. The labeled data in the figure represents the area enclosed by the curve, and the larger the area, the better the performance.

reducing the memory required by three orders of magnitude. Moreover, compared to the recent Queue-based filter, the proposed filter's $F1_d$ has been improved by more than 4.4% with a 77% reduction in memory. It only needs 4.072 kb memory, while the $O(N^2)$ filter needs 10 Mb memory, Liu's filter needs 5 Mb memory, $O(N)$ filter needs 51 kb memory and the Queue-based filter needs $(9+9)\times 968$ bits $=$ 17.424 kb memory.

Appendix C.1.2 *Public dataset experiments*

To make the experiment persuasive, the filters were compared on the public dataset: DVSNOISE20 [10]. It was collected using a DAVIS346 neuromorphic camera, obtaining 16 indoor and outdoor scenes of noisy, real-world data. As shown in Figure C2, two scenes in the dataset DVSNOISE20 were selected. For the sake of observation, the events before and after filtering were accumulated in frames. As shown in Figure C2, the proposed filter achieves a similar denoising performance to the recent Queue-based filter, and it can be seen from the figures of the removed event (i.e. the second-row images of each scene) that our filter preserves the most real events.

Appendix C.2 Results of clustering

Appendix C.2.1 *Clustering for denoising*

In the above experiment, under the low light condition, only using filters to denoise cannot completely remove noise, as shown in Figure C3(a). After passing through a filter and then clustering, the noise around the airplanes is almost completely removed, as shown in Figure C3(b). Moreover, the real events that pass through the clustering algorithm are almost not lost, thanks to the proposed category coverage mechanism. As shown in Table C1, the combination of filtering and clustering has the best comprehensive denoising performance, removing noise while preserving as many real events as possible.

Appendix C.2.2 *Clustering comparison*

To compare event-based clustering methods on ROI extraction, the public traffic dataset [11] was used in the experiment, and we calculated the weighted F1-score as [11] does. Firstly, the dataset was denoised by the proposed filter. For the Queue-based clustering, due to its integration of denoising and clustering, it uses its filter for denoising. For fairness, the initial radius and the step for radius updates of the centroid-and-radii-based clustering [4] are set to 8. For the Queue-based clustering [3] and the proposed clustering, the clustering threshold is also set to 8. And all clustering algorithms can retain up to 8 targets, according to [11]. After clustering, they only retain categories with event numbers greater than 1. The quantitative results are shown in Figure C4. The weighted F1-score of the proposed clustering has improved by an average of 15.7% compared to the recent Queue-based clustering [3].

References

1 B. Son et al., "4.1 A 640×480 dynamic vision sensor with a 9µm pixel and 300Meps address-event representation," 2017 IEEE International Solid-State Circuits Conference (ISSCC), San Francisco, CA, USA, 2017, pp. 66-67, doi: 10.1109/ISSCC.2017.7870263.

2 T. Finateu et al., "5.10 A 1280×720 Back-Illuminated Stacked Temporal Contrast Event-Based Vision Sensor with 4.86µm Pixels, 1.066GEPS Readout, Programmable Event-Rate Controller and Compressive Data-Formatting Pipeline," 2020 IEEE International Solid-State Circuits Conference - (ISSCC), San Francisco, CA, USA, 2020, pp. 112-114, doi: 10.1109/ISSCC19947.2020.9063149.

3 F. Li, Y. Huang, Y. Chen, X. Zeng, W. Li and M. Wang, "Queue-based Spatiotemporal Filter and Clustering for Dynamic Vision Sensor," 2023 IEEE International Symposium on Circuits and Systems (ISCAS), Monterey, CA, USA, 2023, pp. 1-4, doi: 10.1109/ISCAS46773.2023.10181868.

4 A. Linares-Barranco et al., "Low Latency Event-Based Filtering and Feature Extraction for Dynamic Vision Sensors in Real-Time FPGA Applications," in IEEE Access, vol. 7, pp. 134926-134942, 2019.

5 A. Bisulco, F. Cladera, V. Isler and D. D. Lee, "Near-Chip Dynamic Vision Filtering for Low-Bandwidth Pedestrian Detection," 2020 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), Limassol, Cyprus, 2020, pp. 234-239, doi: 10.1109/ISVLSI49217.2020.00050.

6 D. P. Moeys et al., "A Sensitive Dynamic and Active Pixel Vision Sensor for Color or Neural Imaging Applications," in IEEE Transactions on Biomedical Circuits and Systems, vol. 12, no. 1, pp. 123-136, Feb. 2018.

7 H. Liu, C. Brandli, C. Li, S. -C. Liu and T. Delbruck, "Design of a spatiotemporal correlation filter for event-based sensors," 2015 IEEE International Symposium on Circuits and Systems (ISCAS), 2015, pp. 722-725.

8 A. Khodamoradi and R. Kastner, "$O(N)$O(N)-Space Spatiotemporal Filter for Reducing Noise in Neuromorphic Vision Sensors," in IEEE Transactions on Emerging Topics in Computing, vol. 9, no. 1, pp. 15-23, 1 Jan.-March 2021.

9 Delbruck, Tobi. "Frame-free dynamic digital vision." Proceedings of Intl. Symp. on Secure-Life Electronics, Advanced Electronics for Quality Life and Society. Vol. 1. 2008.

10 R. W. Baldwin, M. Almatrafi, V. Asari and K. Hirakawa, "Event Probability Mask (EPM) and Event Denoising Convolutional Neural Network (EDnCNN) for Neuromorphic Cameras," 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2020, pp. 1698-1707.

11 V. Mohan et al., "EBBINNOT: A Hardware-Efficient Hybrid Event-Frame Tracker for Stationary Dynamic Vision Sensors," in IEEE Internet of Things Journal, vol. 9, no. 21, pp. 20902-20917, 1 Nov.1, 2022, doi: 10.1109/JIOT.2022.3178120.