CSEG: A COMPACT STATEFUL EVENT GRAPH REPRESENTATION FOR EFFICIENT VISUAL PROCESSING

Anonymous authors
Paper under double-blind review

ABSTRACT

The sparse, asynchronous output of event cameras is naturally suited to graph representations, yet treating each event as a node makes such graphs computationally expensive. Discarding events reduces this cost but inevitably loses information. Instead, we introduce CSEG, a compact stateful event graph representation that significantly reduces graph size while maintaining competitive detection accuracy. CSEG constructs a compact graph by retaining only a subset of event nodes, selected according to polarity and temporal information. For each dropped event node, local message differences guide the selection of a retained node as its fusion target. Its temporal information is then fused into this node through a temporal state update. We evaluate CSEG on event-based object detection using the Gen1 and DSEC-Det datasets. Compared with existing event graph methods, CSEG achieves competitive detection accuracies while retaining only 42.56% and 44.23% of event nodes on Gen1 and DSEC-Det datasets, respectively. Compared with the event graph baseline, CSEG achieves a 2.6x acceleration in inference, demonstrating its computational efficiency.

1 INTRODUCTION

Event cameras capture visual changes as asynchronous events, each encoding a pixel location, timestamp, and polarity (Gallego et al., 2022). To process these sparse signals with neural networks, events need to be organized into a suitable representation. A common approach is to accumulate events into dense tensors (Gehrig et al., 2019; Zhu et al., 2019). Alternatively, event graphs preserve their sparse and irregular structure by representing events as nodes connected through local spatiotemporal neighborhoods (Schaefer et al., 2022; Gehrig & Scaramuzza, 2024). This representation naturally supports asynchronous computation, since an incoming event only affects a local portion of the graph rather than requiring the entire representation to be recomputed (Li et al., 2021). However, existing event-graph methods typically instantiate each valid event as an explicit node. As a result, the graph size still grows directly with the event rate, increasing the cost of graph construction, message passing, and state maintenance during periods of dense activity. Thus, even when updates are limited to a local part of the graph, retaining every event as a node still incurs substantial computational cost.

A straightforward way to control graph size is to drop a subset of event nodes. Nearby events with the same polarity often describe the same local structure and therefore carry similar spatial information. This spatial redundancy provides an opportunity to reduce the number of nodes. However, their different arrival times still carry temporal information (Lagorce et al., 2017; Baldwin et al., 2023). Simply dropping their nodes also discards this temporal information. Meanwhile, existing event graphs such as AEGNN (Schaefer et al., 2022) and DAGr (Gehrig & Scaramuzza, 2024) use event timestamps to determine processing order and graph connectivity, but temporal information is not explicitly maintained as a persistent node state. Dropping an event node also removes its timestamp and graph connections, so its temporal information is lost from the representation. The key challenge is to retain fewer event nodes while preserving the temporal information of those that are dropped.

Motivated by this distinction, we introduce CSEG, a compact stateful event graph representation. As illustrated in Figure 1, CSEG drops event nodes while preserving their temporal contributions through updates to retained nodes. CSEG first considers nearby retained nodes with the same polarity. It decides whether to retain or drop an event node by measuring how much fusing its temporal information would change a candidate node's temporal feature. For each dropped node, local message

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

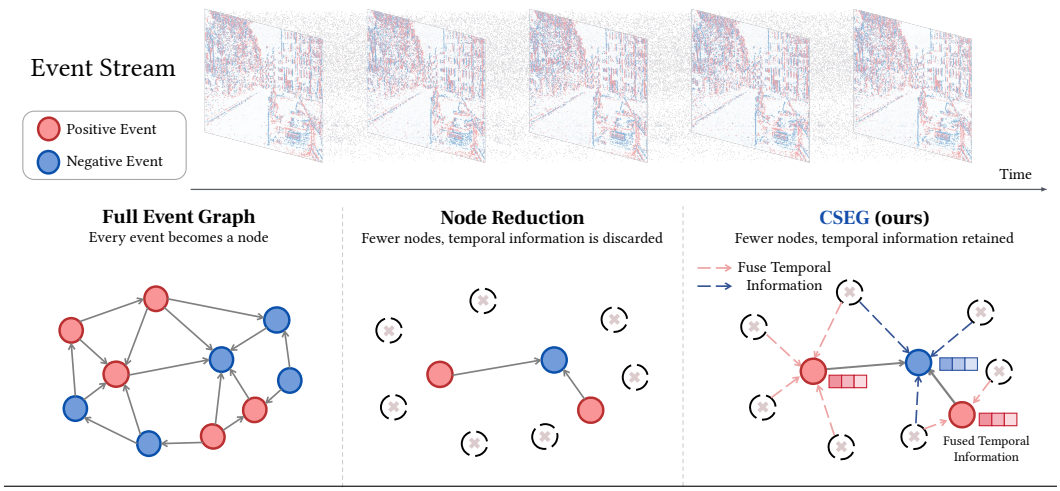


Figure 1: **Comparison of event-graph representations.** A full event graph creates a node for every event. Direct node reduction discards the temporal contributions of dropped event nodes. CSEG reduces graph size by fusing the temporal information of dropped event nodes into retained nodes.

differences guide the selection of a retained node as its fusion target. The temporal information of the dropped event node is then fused into the selected retained node. A single retained node can thus preserve temporal contributions from multiple events. To reduce the cost of fusion target selection, we precompute the terms used to compare local message responses and store them in lookup tables. We also reuse existing graph features to avoid recomputing unaffected nodes during incremental processing.

We evaluate CSEG on event-based object detection using the Gen1 and DSEC-Det datasets. Compared with existing event graph methods, CSEG achieves competitive detection accuracy while retaining only 42.56% and 44.23% of event nodes on the two datasets, respectively. Compared with the event graph baseline DAGr (Gehrig & Scaramuzza, 2024), CSEG achieves a 2.6× speedup in inference. Ablation experiments examine the roles of node retention, temporal information fusion, and fusion target selection.

Our contributions are threefold:

- We introduce CSEG, a compact stateful event graph representation that preserves temporal information from dropped event nodes in retained nodes.
- We propose state-guided node retention and message-guided fusion target selection, with lookup tables and feature reuse supporting efficient incremental processing.
- Experiments show that CSEG achieves competitive detection accuracy while retaining fewer than half of the event nodes and accelerating inference.

2 Related Work

Asynchronous event-graph networks. AEGNN (Schaefer et al., 2022) represents events as evolving spatiotemporal graphs and restricts computation to nodes affected by incoming events. Building on asynchronous graph processing, DAGr (Gehrig & Scaramuzza, 2024) improves efficiency through directed graphs and update pruning, while incorporating image features for detection. ACGR (Li et al., 2025) further explores a unified graph representation for asynchronous frame–event fusion. These advances improve incremental graph computation and multimodal processing. Other event-graph representations, such as EV-VGCNN (Deng et al., 2022), use voxels as graph vertices. CSEG focuses on reducing graph size while preserving temporal information from dropped event nodes.

Temporal event representations. HOTS (Lagorce et al., 2017) encodes recent event timing through decaying time surfaces, while HATS (Sironi et al., 2018) accumulates local event history into averaged time-surface histograms. EST (Gehrig et al., 2019) preserves temporal information in a learnable spatiotemporal tensor, and Matrix-LSTM (Cannici et al., 2020) learns recurrent states from event sequences at each spatial location. These studies demonstrate the importance of preserving temporal

information, but organize it in pixel-based surfaces or regular grids. CSEG instead stores temporal information in retained nodes to form a compact event graph for asynchronous inference.

Event-based vision. Event-based vision has advanced object detection (Perot et al., 2020; Gehrig & Scaramuzza, 2023; Peng et al., 2024), 3D reconstruction (Han et al., 2024), event-to-point-cloud registration (Lin et al., 2023), continuous-time motion estimation (Gehrig et al., 2024), and motion deblurring (Sun et al., 2022). These applications exploit the fine temporal resolution of events to perceive dynamic scenes. Realizing the benefits of asynchronous sensing also requires representations that preserve temporal information while keeping successive updates efficient. CSEG addresses this requirement through a compact, incrementally updated event graph, with object detection serving as the evaluation task.

3 METHOD

CSEG builds a compact stateful asynchronous event graph by retaining selected event nodes and fusing the temporal information of dropped event nodes into retained nodes. Sec. 3.1 introduces the event-graph backbone and SplineConv. Building on this backbone, Sec. 3.2 determines whether to retain or drop each event node, Sec. 3.3 selects a fusion target for each dropped node, and Sec. 3.4 integrates these decisions into the asynchronous graph update and downstream processing. The complete procedure is summarized in Algorithm 1 in Appendix A.

3.1 PRELIMINARIES

Let $e_k = (\mathbf{s}_k, t_k, p_k)$ denote the k-th event, where $\mathbf{s}_k = (x_k, y_k)$ is its spatial location, t_k its timestamp, and $p_k \in \{-1, +1\}$ its polarity. The event stream is represented as a graph in which retained events form nodes, and each newly retained node is connected to previously retained nodes within a local spatiotemporal neighborhood.

Feature propagation on the event graph is performed using SplineConv (Fey et al., 2018). At layer ℓ, each node v has a feature vector $\mathbf{z}_v^{(\ell)}$. Let $\mathcal{N}_v$ denote the set of neighboring nodes whose features are aggregated at v. SplineConv combines the feature of v itself with messages from these neighboring nodes as

$$\mathbf{a}_v^{(\ell+1)} = R_\ell \mathbf{z}_v^{(\ell)} + \sum_{j \in \mathcal{N}_v} W_\ell(\boldsymbol{\xi}_{jv}) \mathbf{z}_j^{(\ell)} + \mathbf{b}_\ell, \tag{1}$$

where $\mathbf{a}_v^{(\ell+1)}$ is the aggregated feature. R_ℓ is the learned linear transformation matrix for the center node at layer ℓ, while $W_\ell(\boldsymbol{\xi}_{jv})$ is the transformation matrix applied to neighboring node j at layer ℓ, conditioned on its normalized relative spatial coordinate $\boldsymbol{\xi}_{jv}$ with respect to node v, and $\mathbf{b}_\ell$ is the bias vector.

3.2 STATE-GUIDED NODE RETENTION

CSEG retains only a subset of event nodes to reduce graph size. The temporal information of dropped event nodes is fused into retained nodes. Let $\pi(k)$ denote the index of the retained node that stores the temporal information of event e_k. If node k is retained, $\pi(k) = k$. Otherwise, $\pi(k) = r$, where r is a retained node that receives its temporal information.

Each retained node r maintains a temporal state q_r:

$$q_r = \sum_{\substack{i \le k \\ \pi(i) = r}} \exp\left(\frac{t_i - t_r}{\tau_q}\right), \tag{2}$$

where t_r is the timestamp of the retained node and τ_q controls the temporal weighting. The state q_r aggregates the temporal contributions of all events represented by r, with later events contributing more strongly relative to the retained node timestamp. The state q_r changes only when additional events are fused into node r. Their total time-decayed contribution at time T can be recovered as $q_r e^{-(T-t_r)/\tau_q}$. A newly retained node therefore starts with $q_r = 1$, and each subsequently represented event contributes one additional exponential term.

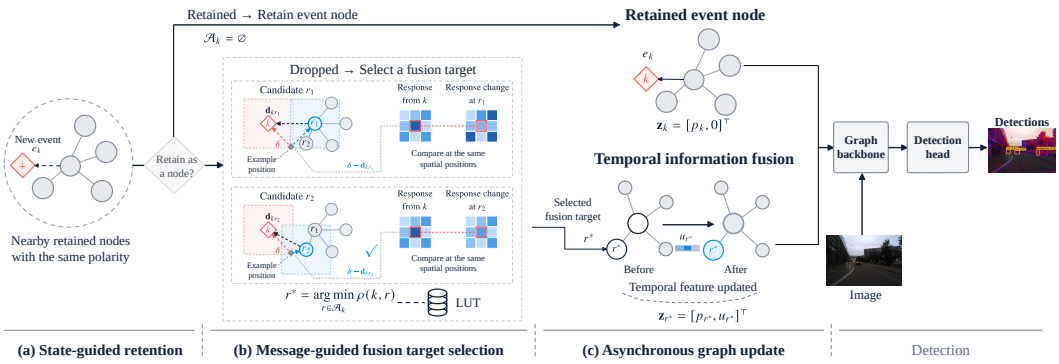


Figure 2: **Overview of CSEG.** (a) Local spatial, temporal, and polarity constraints define candidate nodes. A temporal feature change threshold determines whether to retain or drop each event node. (b) For each dropped event node, fusion target selection uses a lookup table to select the node whose temporal feature update minimizes the message discrepancy relative to retaining the event node. (c) The graph network processes retained event nodes and temporal feature updates of fusion targets asynchronously, and its features are fused with image features for object detection.

Since q_r in Eq. (2) increases as more events are represented by r, we map it to a bounded temporal feature u_r for use as a graph-network input

$$u_r = f(q_r), \qquad f(q) = \frac{\log q}{1 + \log q}, \tag{3}$$

where $u_r \in [0, 1)$ and $u_r = 0$ for a newly retained node.

Candidate selection. For a new event e_k, we first identify nearby retained nodes that can receive its temporal information. We require these nodes to have the same polarity as e_k and lie within spatial and temporal limits. As illustrated in Figure 2, these constraints define the candidate set

$$\mathcal{C}_k = \{r \in \mathcal{V}_{k-1} \mid p_r = p_k,\ \|\mathbf{s}_k - \mathbf{s}_r\|_\infty \le d_s,\ 0 \le t_k - t_r \le d_t\}, \tag{4}$$

where $\mathcal{V}_{k-1}$ is the set of previously retained nodes, and d_s and d_t specify the spatial and temporal limits.

Retention criterion. We then determine whether the temporal information of e_k can be fused into one of these candidates. For each $r \in \mathcal{C}_k$, the fusion would change its temporal feature by

$$\Delta u(k, r) = f\left(q_r + \exp\left(\frac{t_k - t_r}{\tau_q}\right)\right) - f(q_r). \tag{5}$$

We allow this update when $\Delta u(k, r) \le \theta$, giving the set

$$\mathcal{A}_k = \{r \in \mathcal{C}_k \mid \Delta u(k, r) \le \theta\}. \tag{6}$$

If $\mathcal{A}_k$ is empty, event node k is retained. Otherwise, it is dropped, and its temporal information is fused into one of the nodes in $\mathcal{A}_k$. Section 3.3 describes how this node is selected.

3.3 Message-Guided Fusion Target Selection

When an event node is dropped, CSEG selects a fusion target from $\mathcal{A}_k$. This set may contain multiple retained nodes. We seek a retained node as the fusion target whose temporal feature update produces a local message response as close as possible to that of retaining the original event node. To guide this selection, we define the message discrepancy $\rho(k, r)$ between the response of retaining node k and the response change caused by updating candidate r. The candidate with the smallest discrepancy is selected as the fusion target.

At the first SplineConv layer in Eq. (1), each retained node r carries the feature $\mathbf{z}_r = [p_r, u_r]^\top$. As illustrated in Figure 2, we compare the local message response produced by retaining node k with the change in response caused by updating a candidate fusion target $r \in \mathcal{A}_k$. If node k is retained, its temporal feature is initialized to zero, giving

$$\mathbf{m}_{\text{keep}}(k, \boldsymbol{\delta}) = W_1(\boldsymbol{\xi}(\boldsymbol{\delta})) \begin{bmatrix} p_k \\ 0 \end{bmatrix}. \tag{7}$$

If node k is dropped and its temporal contribution is fused into r, only the temporal feature of r changes by $\Delta u(k, r)$ in Eq. (5), giving the following change in local message response

$$\mathbf{m}_{\text{update}}(k, r, \boldsymbol{\delta}) = W_1(\boldsymbol{\xi}(\boldsymbol{\delta} - \mathbf{d}_{kr})) \begin{bmatrix} 0 \\ \Delta u(k, r) \end{bmatrix}. \tag{8}$$

Here, $\boldsymbol{\delta}$ is a spatial offset and $\mathbf{d}_{kr} = \mathbf{s}_k - \mathbf{s}_r$ is the displacement from r to k. The offsets $\boldsymbol{\delta}$ and $\boldsymbol{\delta} - \mathbf{d}_{kr}$, normalized by $\boldsymbol{\xi}$ as in Eq. (1), allow the two responses to be compared at the same spatial position.

We measure the difference between these two responses as

$$\rho(k, r)^2 = \sum_{\boldsymbol{\delta}} \|\mathbf{m}_{\text{keep}}(k, \boldsymbol{\delta}) - \mathbf{m}_{\text{update}}(k, r, \boldsymbol{\delta})\|_2^2, \tag{9}$$

where the sum is taken over the local spatial offsets used to evaluate both responses. A smaller $\rho(k, r)$ indicates closer local message responses. The fusion target is selected as

$$r^* = \arg\min_{r \in \mathcal{A}_k} \rho(k, r). \tag{10}$$

To accelerate fusion target selection, we precompute the kernel-dependent terms of the message discrepancy in Eq. (9) and store them in a lookup table. This avoids explicitly evaluating the local message responses for every candidate. The spatial evaluation details and lookup-table derivation are provided in Appendix B.

3.4 Asynchronous Graph Update

CSEG processes each new event e_k as it arrives, while maintaining the graph structure and intermediate activations produced by previously processed events (Schaefer et al., 2022; Gehrig & Scaramuzza, 2024). This persistent graph state enables event-wise asynchronous updates.

Retained event node. If node k is retained, it is added to the graph with feature $\mathbf{z}_k = [p_k, 0]^\top$, as illustrated in Figure 2. The existing graph is first searched for neighbors of e_k, and the resulting neighbor set is denoted by $\mathcal{N}_k$. The activation of node k at layer $\ell + 1$ is computed as

$$\mathbf{a}_k^{(\ell+1)} = R_\ell \mathbf{z}_k^{(\ell)} + \sum_{j \in \mathcal{N}_k} W_\ell(\boldsymbol{\xi}_{jk}) \mathbf{z}_j^{(\ell)} + \mathbf{b}_\ell. \tag{11}$$

The new node k aggregates information from its existing neighbors. Its resulting feature $\mathbf{z}_k^{(\ell+1)}$ is passed to the next graph layer. Subsequent computation updates the affected nodes and reuses the features of unaffected nodes.

Dropped event node. If event node k is dropped, its temporal contribution updates the selected fusion target r^* from Eq. (10), as illustrated in Figure 2. This produces the input feature change

$$\Delta \mathbf{z}_{r^*}^{(0)} = \begin{bmatrix} 0 \\ \Delta u(k, r^*) \end{bmatrix}. \tag{12}$$

Since the graph topology and node positions remain unchanged, only the cached convolution terms that depend on r^* need to be updated. The contribution from the node's own feature is updated by

$$\mathbf{a}_{r^*}^{(\ell+1)} = \mathbf{a}_{r^*}^{(\ell+1)} + R_\ell \Delta \mathbf{z}_{r^*}^{(\ell)}. \tag{13}$$

Updating r^* also changes the messages it sends to its neighbors. For each neighboring node v receiving such a message, the cached activation is updated by

$$\mathbf{a}_v^{(\ell+1)} = \mathbf{a}_v^{(\ell+1)} + W_\ell(\boldsymbol{\xi}_{r^* v}) \Delta \mathbf{z}_{r^*}^{(\ell)}. \tag{14}$$

The resulting feature changes are propagated recursively to the subsequent layers, while all unaffected activations are reused.

For DSEC-Det, the graph features are fused with image features following DAGr (Gehrig & Scaramuzza, 2024), as shown in Figure 2. Gen1 uses event features only. The detection head (Ge et al., 2021) predicts object classes and bounding boxes from the resulting features. The trainable network parameters are optimized using the detection loss.

4 EXPERIMENTS

We evaluate CSEG on event-based object detection to assess its effectiveness in reducing graph size while maintaining detection accuracy. We compare detection performance and node retention on the Gen1 and DSEC-Det datasets, and measure the computational cost and latency of asynchronous inference.

4.1 EXPERIMENTAL SETUP

Implementation details. The CSEG graph backbone comprises five residual stages with channel widths of $(16, 64, 64, 64, 64)$, each containing two spline graph convolutions with batch normalization (Ioffe & Szegedy, 2015) and ReLU activation. We use two-dimensional spline kernels with a 5×5 control-point grid and limit each node to 16 incoming neighbors. We train the graph network synchronously on batched event graphs and convert it to asynchronous inference using the learned weights. During inference, intermediate node features are cached, and newly retained nodes or temporal feature changes of fusion targets trigger incremental updates to the affected graph features. Spatial spline-convolution weights are precomputed in lookup tables (LUTs) indexed by relative node offsets. We set $d_s = 1$ pixel, $d_t = 3.5\,\mathrm{ms}$, $\tau_q = 50\,\mathrm{ms}$, and $\theta \approx 0.414$. Image fusion and the detection head follow DAGr (Gehrig & Scaramuzza, 2024).

Training details. For Gen1, we use event-only inputs at 304×240 resolution, with each sample containing the $50\,\mathrm{ms}$ of events preceding the detection timestamp. For DSEC-Det, we pair each image with the subsequent $50\,\mathrm{ms}$ of events at 320×215 resolution. We initialize the DSEC-Det model from pretrained DAGr weights, set the additional temporal-input weights to zero, and train for 61 epochs using AdamW (Loshchilov & Hutter, 2019) with a learning rate of 2×10^{-4} and a batch size of 64.

Datasets and evaluation. We evaluate CSEG on the Gen1 (de Tournemire et al., 2020) and DSEC-Det datasets (Gehrig et al., 2021; Gehrig & Scaramuzza, 2024). We compare detection accuracy using mean average precision (mAP) (Lin et al., 2014), computational cost using GFLOPs, and graph size using the percentage of retained event nodes.

4.2 DETECTION PERFORMANCE AND GRAPH COMPRESSION

Table 1: **Object detection performance and computational cost on Gen1 and DSEC-Det.**

Method	Rep.	Det. head	Gen1			DSEC-Det		
			Retained (%)	mAP	GFLOPs	Retained (%)	mAP	GFLOPs
Inception+SSD	Frame	SSD	100%	30.1	–	100%	18.4	–
Events+YOLOv3	Frame	YOLOv3	100%	31.2	–	100%	28.7	–
SAST (2024)	Voxel	YOLOX	100%	**48.2**	90.25	100%	38.1	258.3
RVT (2023)	Voxel	YOLOX	100%	<u>47.2</u>	104.49	100%	38.4	293.6
DAGr (2024)	Graph	YOLOX	100%	32.1	4.96	100%	<u>41.9</u>	13.93
CSEG (Ours)	Graph	YOLOX	42.56%	38.7	3.44	44.23%	**42.17**	3.46

Detection accuracy. The first four methods in Table 1 process event frames or voxel representations synchronously, while DAGr and CSEG support asynchronous graph inference. CSEG retains fewer than half of the event nodes on both datasets. On Gen1, it achieves higher mAP than DAGr, but lower mAP than the Transformer-based RVT (Gehrig & Scaramuzza, 2023) and SAST (Peng et al., 2024), with substantially fewer GFLOPs. On DSEC-Det, CSEG achieves the highest mAP among the compared methods. On Gen1, CSEG achieves 38.7% mAP with 42.56% of the event nodes and 3.44 GFLOPs. On DSEC-Det, CSEG achieves 42.17% mAP with 44.23% of the event nodes and 3.46 GFLOPs.

4.3 Qualitative Detection Results

Figure 3 qualitatively compares DAGr and CSEG under identical event inputs. CSEG detects more ground-truth objects and produces fewer false positives than DAGr, demonstrating improved detection accuracy.

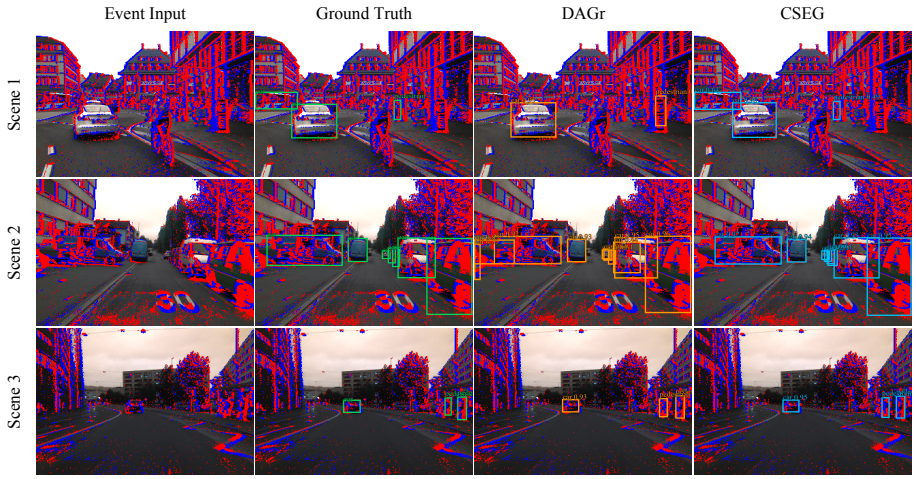


Figure 3: **Qualitative detection results.** CSEG improves detection accuracy over DAGr.

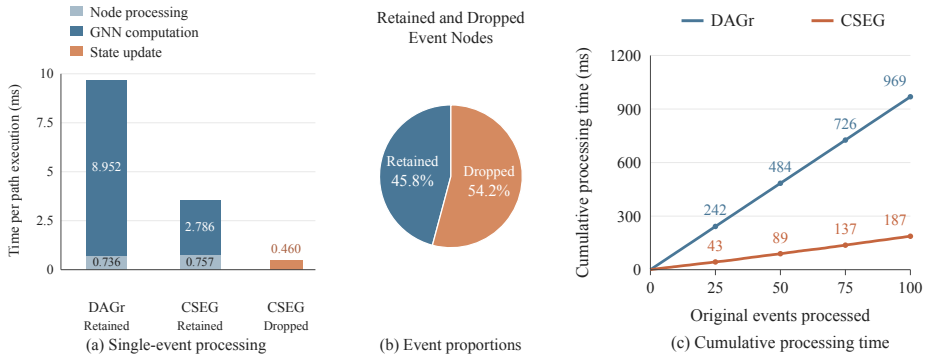


Figure 4: **Single-event processing time, event proportions, and cumulative processing time.**

4.4 Computational Efficiency

We evaluate the computational efficiency of both methods on an NVIDIA L40 using 24 DSEC test windows and cached image features. We first consider a system that continuously receives events and queries detection results at fixed intervals in event time. We therefore measure the cost of processing event batches, with each batch containing the events arriving within a 1 ms interval and triggering one detection output. As shown in Table 2, CSEG reduces computation from 565.89 to 342.93 MFLOPs and mean processing time from 13.53 to 5.20 ms per batch compared with the optimized DAGr implementation, corresponding to a 2.6× speedup. We also report the 95th-percentile (P95) processing time and peak memory usage. CSEG lowers both measures.

Table 2: **Asynchronous inference cost with 1 ms event batches.**

Method	Event MFLOPs (per batch) ↓	Mean processing time (ms) ↓	P95 processing time (ms) ↓	Peak memory (GiB) ↓
DAGr	565.89	13.53	15.59	4.67
CSEG	**342.93**	**5.20**	**6.11**	**4.47**

We then measure the processing time of individual events. Figure 4(a) compares the costs of the different processing paths. DAGr retains every event as a node, requiring 0.736 ms for node processing and 8.952 ms for GNN computation, totaling 9.688 ms per event. For CSEG, processing a retained

event requires 0.757 ms for node processing and 2.786 ms for GNN computation, totaling 3.543 ms. Processing a dropped event requires 0.460 ms for fusion target selection and temporal information fusion using LUTs. Figure 4(b) shows that 45.8% of events are retained as nodes and require 3.543 ms each, while the remaining 54.2% require only 0.460 ms each for fusion target selection and temporal information fusion. Figure 4(c) shows that CSEG consistently reduces cumulative processing time compared with DAGr.

5 Ablation Experiments

To validate the effectiveness of CSEG, we conduct ablation studies on the DSEC-Det dataset covering node retention, temporal information fusion, and fusion target selection.

5.1 Component Ablations

We evaluate the effectiveness of node retention, temporal information fusion, and fusion target selection in Table 3, with full-event DAGr as a reference. A checkmark (✓) denotes the corresponding CSEG component. A cross (×) denotes random selection for nodes or fusion targets, or disabled temporal information fusion. A dash (—) indicates that a component is not applicable. Fusion target selection has no effect when temporal information fusion is disabled.

Random node selection uses a retention rate of 44.23%, matching the full CSEG model. Random fusion target selection samples uniformly from retained nodes in $\mathcal{A}_k$. The Random variants share the DAGr initialization, training order, and seed.

Table 3: **Component ablations on the DSEC-Det dataset.**

Method	Node retention	Temporal information fusion	Fusion target selection	Retained (%) ↓	mAP ↑
DAGr	—	—	—	100.00	41.90
Random	×	×	—	44.23	39.69
Random	×	✓	×	44.23	40.27
Random	×	✓	✓	44.23	40.36
CSEG	✓	×	—	44.23	41.31
CSEG	✓	✓	×	43.63	41.70
CSEG	✓	✓	✓	**44.23**	**42.17**

Table 3 shows that temporal information fusion raises Random's mAP from 39.69% to 40.27%, with message-guided fusion target selection reaching 40.36% at the same 44.23% retention. Full CSEG achieves 42.17% mAP, while the variant without temporal information fusion achieves 41.31% at the same 44.23% retention.

5.2 Preservation of Temporal Activity

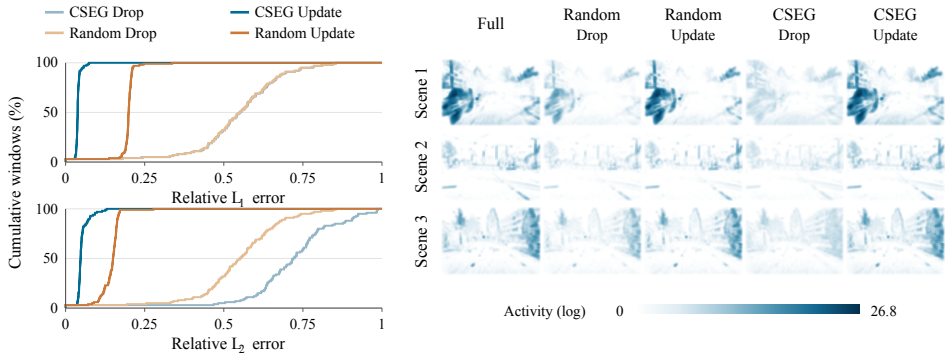
We evaluate temporal activity preservation by comparing the time-decayed activity maps represented by retained nodes with those of the full event stream. We compare Random and CSEG at matched node budgets using relative L_1 and L_2 errors after spatial smoothing on DSEC validation windows. In Table 4, a checkmark (✓) enables temporal information fusion using the temporal contributions of dropped event nodes, while a cross (×) includes only the contributions of retained events. Appendix C.1 defines the activity maps and error metrics.

Table 4: **Temporal activity errors at matched node budgets.**

Method	Temporal information fusion	Relative L_1 ↓	Relative L_2 ↓
Random	×	0.54039	0.54161
Random	✓	0.19454	0.14150
CSEG	×	0.54224	0.70914
CSEG	✓	**0.03840**	**0.05155**

As shown in Table 4, temporal information fusion reduces CSEG's relative L_1 error from 0.542 to 0.038 and its relative L_2 error from 0.709 to 0.052. With temporal information fusion enabled, CSEG also achieves lower temporal activity errors than Random at the same node budget. Without temporal information fusion, CSEG does not outperform Random. Including the temporal contributions of dropped event nodes substantially improves temporal activity preservation in this setting.

In Figure 5, "Drop" excludes the temporal information of dropped event nodes, while "Update" includes it through temporal information fusion. Figure 5(a) summarizes the reconstruction errors across the evaluated event windows. We compute a relative L_1 error and a relative L_2 error for each window. In each plot, the horizontal axis gives an error value, and the vertical axis shows the percentage of windows whose error is no greater than that value. A curve that rises earlier therefore indicates that more windows have small reconstruction errors. With temporal information fusion, CSEG achieves low errors in most windows. Figure 5(b) provides visual examples. The activity maps reconstructed by CSEG remain close to the full-event reference in both spatial patterns and intensity. Without this fusion, dropping event nodes reduces the reconstructed activity. Together, these results show that fusing temporal information into retained nodes helps preserve the temporal contributions of dropped event nodes.



(a) Temporal-activity errors (b) Temporal activity

Figure 5: **Temporal activity preservation measured by (a) cumulative error distributions and (b) activity maps with and without temporal information fusion.**

6 Conclusion and Limitations

In this paper, we propose CSEG, a compact stateful event-graph representation that preserves aggregate temporal contributions through node states while reducing the cost of asynchronous graph operations for lower-latency inference. CSEG combines state-guided node retention with message-guided fusion target selection. Temporal information from dropped event nodes is fused into the selected retained nodes. These decisions rely only on the current event and previously retained nodes, supporting causal and incremental graph processing. Experiments on event-based object detection demonstrate that CSEG reduces graph size and computational cost while maintaining competitive detection accuracy. Further ablations highlight the effectiveness of node retention and temporal information fusion, while asynchronous evaluations show reduced inference latency.

Limitations. CSEG preserves an aggregate of the temporal contributions of dropped event nodes. The state does not retain the full timestamp distribution, and graph compression still loses some graph-structure information. The message discrepancy guides fusion target selection and does not guarantee that all this information is preserved. CSEG relies on local spatiotemporal redundancy to reduce graph size, so its compression ratio and computational savings depend on the event stream. Achieving real-time deployment may require dedicated graph-processing accelerators and execution architectures optimized for sparse memory access and incremental updates. A pipelined implementation could also be explored, but this is beyond the scope of this work.

AI USE STATEMENT

AI tools assisted with experiment design, method implementation, data cleaning, result analysis, figure preparation, and manuscript drafting and editing. The authors take responsibility for the final content, results, and claims of this work.

REFERENCES

R. Wes Baldwin, Ruixu Liu, Mohammed Almatrafi, Vijayan Asari, and Keigo Hirakawa. Time-ordered recent event (TORE) volumes for event cameras. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(2):2519–2532, 2023. doi: 10.1109/TPAMI.2022.3172212.

Marco Cannici, Marco Ciccone, Andrea Romanoni, and Matteo Matteucci. A differentiable recurrent surface for asynchronous event-based data. In *Proceedings of the European Conference on Computer Vision*, pp. 136–152, 2020.

Pierre de Tournemire, Davide Nitti, Etienne Perot, Davide Migliore, and Amos Sironi. A large scale event-based detection dataset for automotive, 2020. URL https://arxiv.org/abs/2001.08499.

Yongjian Deng, Hao Chen, Hai Liu, and Youfu Li. A voxel graph CNN for object classification with event cameras. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1172–1181, 2022. URL https://openaccess.thecvf.com/content/CVPR2022/html/Deng_A_Voxel_Graph_CNN_for_Object_Classification_With_Event_Cameras_CVPR_2022_paper.html.

Matthias Fey, Jan Eric Lenssen, Frank Weichert, and Heinrich M"uller. SplineCNN: Fast geometric deep learning with continuous B-spline kernels. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 869–877, 2018.

Guillermo Gallego, Tobi Delbr"uck, Garrick Orchard, Chiara Bartolozzi, Brian Taba, Andrea Censi, Stefan Leutenegger, Andrew J. Davison, J"org Conradt, Kostas Daniilidis, and Davide Scaramuzza. Event-based vision: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(1):154–180, 2022. doi: 10.1109/TPAMI.2020.3008413.

Zheng Ge, Songtao Liu, Feng Wang, Zeming Li, and Jian Sun. YOLOX: Exceeding YOLO series in 2021, 2021. URL https://arxiv.org/abs/2107.08430.

Daniel Gehrig and Davide Scaramuzza. Low-latency automotive vision with event cameras. *Nature*, 629(8014):1034–1040, 2024. doi: 10.1038/s41586-024-07409-w.

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

Mathias Gehrig and Davide Scaramuzza. Recurrent vision transformers for object detection with event cameras. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 13884–13893, 2023.

Mathias Gehrig, Willem Aarents, Daniel Gehrig, and Davide Scaramuzza. DSEC: A stereo event camera dataset for driving scenarios. *IEEE Robotics and Automation Letters*, 6(3):4947–4954, 2021. doi: 10.1109/LRA.2021.3068942.

Mathias Gehrig, Manasi Muglikar, and Davide Scaramuzza. Dense continuous-time optical flow from event cameras. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 46(7): 4736–4746, 2024. doi: 10.1109/TPAMI.2024.3361671.

Haiqian Han, Jianing Li, Henglu Wei, and Xiangyang Ji. Event-3DGS: Event-based 3d reconstruction using 3d gaussian splatting. In *Advances in Neural Information Processing Systems*, volume 37, 2024.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the International Conference on Machine Learning*, pp. 448–456, 2015. URL https://proceedings.mlr.press/v37/ioffe15.html.

Xavier Lagorce, Garrick Orchard, Francesco Galluppi, Bertram E. Shi, and Ryad B. Benosman. HOTS: A hierarchy of event-based time-surfaces for pattern recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(7):1346–1359, 2017. doi: 10.1109/TPAMI.2016.2574707.

Dianze Li, Jianing Li, Xu Liu, Xiaopeng Fan, and Yonghong Tian. Asynchronous collaborative graph representation for frames and events. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1655–1666, 2025. URL https://openaccess.thecvf.com/content/CVPR2025/papers/Li_Asynchronous_Collaborative_Graph_Representation_for_Frames_and_Events_CVPR_2025_paper.pdf.

Yijin Li, Han Zhou, Bangbang Yang, Ye Zhang, Zhaopeng Cui, Hujun Bao, and Guofeng Zhang. Graph-based asynchronous event processing for rapid object recognition. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 934–943, 2021. URL https://openaccess.thecvf.com/content/ICCV2021/html/Li_Graph-Based_Asynchronous_Event_Processing_for_Rapid_Object_Recognition_ICCV_2021_paper.html.

Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: Common objects in context. In *Proceedings of the European Conference on Computer Vision*, pp. 740–755, 2014. doi: 10.1007/978-3-319-10602-1_48.

Xiuhong Lin, Changjie Qiu, Zhipeng Cai, Siqi Shen, Yu Zang, Weiquan Liu, Xuesheng Bian, Matthias Müller, and Cheng Wang. E2PNet: Event to point cloud registration with spatio-temporal representation learning. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/3a2d1bf9bc0a9794cf82c1341a7a75e6-Abstract-Conference.html.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

Dongyue Lu, Lingdong Kong, Gim Hee Lee, Camille Simon Chane, and Wei Tsang Ooi. FlexEvent: Towards flexible event-frame object detection at varying operational frequencies. In *Advances in Neural Information Processing Systems*, volume 38, 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/8064e4ebbcbe594628887b420956d8c3-Abstract-Conference.html.

Yansong Peng, Hebei Li, Yueyi Zhang, Xiaoyan Sun, and Feng Wu. Scene adaptive sparse transformer for event-based object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024.

Etienne Perot, Pierre de Tournemire, Davide Nitti, Jonathan Masci, and Amos Sironi. Learning to detect objects with a 1 megapixel event camera. In *Advances in Neural Information Processing Systems*, volume 33, 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/hash/c213877427b46fa96cff6c39e837ccee-Abstract.html.

Simon Schaefer, Daniel Gehrig, and Davide Scaramuzza. AEGNN: Asynchronous event-based graph neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12371–12381, 2022.

Amos Sironi, Manuele Brambilla, Nicolas Bourdis, Xavier Lagorce, and Ryad Benosman. HATS: Histograms of averaged time surfaces for robust event-based object classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1731–1740, 2018.

Lei Sun, Christos Sakaridis, Jingyun Liang, Qi Jiang, Kailun Yang, Peng Sun, Yaozu Ye, Kaiwei Wang, and Luc Van Gool. Event-based fusion for motion deblurring with cross-modal attention. In *Proceedings of the European Conference on Computer Vision*, 2022. URL https://www.ecva.net/papers/eccv_2022/papers_ECCV/html/3239_ECCV_2022_paper.php.

Alex Zihao Zhu, Liangzhe Yuan, Kenneth Chaney, and Kostas Daniilidis. Unsupervised event-based learning of optical flow, depth, and egomotion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019.

APPENDIX

Section A summarizes the complete CSEG procedure and explains when node states and graph features are updated. Section B describes the spatial comparison of local message responses and derives the lookup table computation. Section C describes the temporal activity evaluation, the implementation used to accelerate inference, and the effect of different node retention settings. Section D describes the code provided for review and the planned public release.

A OVERALL ALGORITHM

Algorithm 1 summarizes how CSEG processes events in temporal order. Each event node is either retained or dropped. If it is dropped, its temporal information is fused into a selected retained node, called the fusion target. The graph network uses these changes at the next scheduled execution.

Algorithm 1 CSEG event processing and graph updates

Input: Ordered events $\{e_k\}$, trained network and LUT, $d_s, d_t, \tau_q, \theta, T_{\mathrm{ref}}$, execution schedule, and image features when used
Output: Retained graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and queried detections

Initialize $\mathcal{V}, \mathcal{E} \leftarrow \varnothing$, cached features, and pending changes
for each event $e_k = (\mathbf{s}_k, t_k, p_k)$ **do**
  Find candidate nodes $\mathcal{C}_k$ using Eq. (4)
  Compute $\Delta u(k, r)$ for each $r \in \mathcal{C}_k$ using Eq. (5)
  $\mathcal{A}_k \leftarrow \{r \in \mathcal{C}_k : \Delta u(k, r) \le \theta,\ t_k - t_r^{\mathrm{last}} \ge T_{\mathrm{ref}}\}$
  if $\mathcal{A}_k = \varnothing$ **then**
    Search the existing graph for neighbors $\mathcal{N}_k$ of e_k
    Add node k and its graph connections to $\mathcal{G}$
    $q_k \leftarrow 1$, $u_k \leftarrow 0$, $t_k^{\mathrm{last}} \leftarrow -\infty$
    Record the new node for the next graph update
  else
    Compute $\rho(k, r)$ for each $r \in \mathcal{A}_k$ using the LUT in Eq. (S3)
    $r^* \leftarrow \arg\min_{r \in \mathcal{A}_k} \rho(k, r)$
    $q_{r^*} \leftarrow q_{r^*} + \exp((t_k - t_{r^*})/\tau_q)$
    $u_{r^*} \leftarrow f(q_{r^*})$, $t_{r^*}^{\mathrm{last}} \leftarrow t_k$
    Record the changed temporal feature of the fusion target for the next graph update
  end if
  if graph execution is scheduled **then**
    Process all pending changes and update affected graph features
    Reuse unaffected features and clear the processed changes
  end if
  if a detection output is requested **then**
    Apply and clear any pending graph changes, then run the detection head
  end if
end for

Candidate scoring does not modify node states. Only the selected fusion target is updated, while its position and timestamp remain fixed. The set $\mathcal{C}_k$ contains nearby retained nodes with the same polarity. The set $\mathcal{A}_k$ contains those that satisfy the update conditions and can be selected as fusion targets. The refractory interval T_{ref} limits repeated updates when Δu becomes small, and t_r^{last} records the last accepted update time. The main-paper detection results use $T_{\mathrm{ref}} = 0$, which recovers the original rule. $\mathcal{N}_k$ contains the graph neighbors of a newly retained node. If several candidates have the same score, we choose the one with the smallest event index.

The execution schedule follows the evaluation setting. With fixed-interval queries, the graph network processes the pending changes and produces a detection output after each $1\ \mathrm{ms}$ event batch. In the event-wise experiment, retaining an event node triggers graph computation and detection. When an event node is dropped, temporal information fusion updates the selected node while the latest detection output is reused. The resulting feature change is processed when the next event node is retained.

B FUSION TARGET SELECTION

Comparing local responses. We compare the two local message responses at the same spatial positions. Here, $\mathbf{s}_k = (x_k, y_k)$ and $\mathbf{s}_r = (x_r, y_r)$ are the spatial coordinates of event k and candidate node r, respectively. For a position $\mathbf{x}$, let $\boldsymbol{\delta} = \mathbf{s}_k - \mathbf{x}$ be the offset from this position to event k. The corresponding offset for candidate node r is $\mathbf{s}_r - \mathbf{x} = \boldsymbol{\delta} - \mathbf{d}_{kr}$, where $\mathbf{d}_{kr} = \mathbf{s}_k - \mathbf{s}_r$ is the displacement from r to k.

Let $P(\boldsymbol{\delta})$ and $U(\boldsymbol{\delta})$ be the output vectors obtained from the polarity and temporal feature columns of the first SplineConv kernel, using the coordinate normalization in Eq. (1). We evaluate these vectors at integer pixel offsets within $\{-4, \dots, 4\}^2$ and set them to zero outside this region. The discrepancy in Eq. (9) becomes

$$\rho(k, r)^2 = \sum_{\boldsymbol{\delta}} \left\| p_k P(\boldsymbol{\delta}) - \Delta u(k, r) U(\boldsymbol{\delta} - \mathbf{d}_{kr}) \right\|_2^2 . \tag{S1}$$

At each position, we compare the two output vectors, then sum their squared differences over all positions covered by either response. These positions do not need to contain graph nodes. The score compares the message response from the event polarity with the response change from updating a candidate's temporal feature. It does not measure the full network change caused by dropping an event node.

Lookup table computation. Expanding Eq. (S1) gives

$$\begin{aligned} \rho(k, r)^2 = p_k^2 \sum_{\boldsymbol{\delta}} \|P(\boldsymbol{\delta})\|_2^2 + \Delta u(k, r)^2 \sum_{\boldsymbol{\delta}} \|U(\boldsymbol{\delta} - \mathbf{d}_{kr})\|_2^2 \\ - 2 p_k \Delta u(k, r) \sum_{\boldsymbol{\delta}} \langle P(\boldsymbol{\delta}), U(\boldsymbol{\delta} - \mathbf{d}_{kr}) \rangle . \end{aligned} \tag{S2}$$

Since $p_k^2 = 1$ and shifting the response does not change its summed squared norm, we obtain

$$\rho(k, r)^2 = E_p + \Delta u(k, r)^2 E_u - 2 p_k \Delta u(k, r) C(\mathbf{d}_{kr}), \tag{S3}$$

where $E_p = \sum_{\boldsymbol{\delta}} \|P(\boldsymbol{\delta})\|_2^2$, $E_u = \sum_{\boldsymbol{\delta}} \|U(\boldsymbol{\delta})\|_2^2$, and $C(\mathbf{d}_{kr}) = \sum_{\boldsymbol{\delta}} \langle P(\boldsymbol{\delta}), U(\boldsymbol{\delta} - \mathbf{d}_{kr}) \rangle$. These terms depend only on the kernel weights and relative displacement, so they can be precomputed. Fusion target selection then uses the event polarity, temporal feature change, and one table lookup for each candidate.

LUT use during training. At the start of each epoch, we compute E_p, E_u, and $C(\mathbf{d}_{kr})$ from the current first-layer weights. The table stays fixed within that epoch and guides fusion target selection when forming training graphs. The network is trained on these graphs using the detection loss. Gradients update the network weights but do not pass through the discrete node retention or fusion target selection decisions. The next epoch uses a new table computed from the updated weights. For validation and testing, the table is computed from the exponential moving average (EMA) weights of the evaluated model.

Graph updates and pooling. Temporal information fusion leaves the retained node positions and timestamps fixed. Graph connections use a separate four-pixel spatial neighborhood, a $10\,\mathrm{ms}$ time range, and at most 16 incoming neighbors. The first pooling stage weights event features by node recency before taking a channelwise maximum:

$$\mathbf{F}_c = \max_{r \in c} \left\{ e^{-(T_{\mathcal{V}} - t_r)/\tau_q} \mathbf{F}_r \right\}, \qquad T_{\mathcal{V}} = \max_{r \in \mathcal{V}} t_r, \tag{S4}$$

where c is a pooling cell and $\mathbf{F}_r$ is the event feature of node r. Image features are pooled without this temporal weight. After pooling, connected nodes have edges in both directions. Feature changes can therefore affect existing nodes in later layers, while unaffected features are reused. On DSEC-Det, image fusion and the detection head follow DAGr (Gehrig & Scaramuzza, 2024).

C Experimental Details

C.1 Temporal Activity Evaluation

We reconstruct time-decayed activity maps from the temporal states of retained nodes and compare them with the full event stream. For a window with N valid events and final timestamp $T = t_N$, the reference activity at position $\mathbf{s}$ and polarity p is

$$A(\mathbf{s}, p) = \sum_{\substack{1 \le i \le N \\ \mathbf{s}_i = \mathbf{s},\ p_i = p}} e^{-(T - t_i)/\tau_q}. \tag{S5}$$

With temporal information fusion, each retained node contributes the time-decayed value recovered from its temporal state. Without fusion, only the retained event contributes:

$$\begin{aligned} A_{\text{update}}(\mathbf{s}, p) &= \sum_{\substack{r \in \mathcal{V}_N \\ \mathbf{s}_r = \mathbf{s},\ p_r = p}} q_r e^{-(T - t_r)/\tau_q}, \\ A_{\text{retain}}(\mathbf{s}, p) &= \sum_{\substack{r \in \mathcal{V}_N \\ \mathbf{s}_r = \mathbf{s},\ p_r = p}} e^{-(T - t_r)/\tau_q}. \end{aligned} \tag{S6}$$

We apply a 9×9 mean filter $\mathcal{B}$ to each polarity map, using zero padding and keeping the full filtered map. For either retained-node activity map $\widehat{A}$, the relative errors are

$$E_1 = \frac{\left\| \mathcal{B}\widehat{A} - \mathcal{B}A \right\|_1}{\|\mathcal{B}A\|_1 + \varepsilon}, \qquad E_2 = \frac{\left\| \mathcal{B}\widehat{A} - \mathcal{B}A \right\|_2}{\|\mathcal{B}A\|_2 + \varepsilon}, \qquad \varepsilon = 10^{-12}. \tag{S7}$$

Both polarities contribute to each norm. These errors measure the difference between spatially smoothed activity maps.

We use 100 fixed DSEC validation windows. Random selects exactly as many nodes as CSEG in each window, then assigns events to earlier nodes with the same polarity within one pixel and $3.5\,\mathrm{ms}$. It uses the same message discrepancy for fusion target selection without applying the temporal feature change threshold. Dropped event nodes without an eligible fusion target make no temporal contribution to the reduced graph. Exact count matching uses the full window, while fusion target selection uses only earlier retained nodes. Within each node selection policy, the two activity maps use the same retained nodes and fusion targets.

C.2 Efficient Implementation

C++ event processing. Candidate search, node retention, fusion target selection, and temporal information fusion are implemented in C++. This avoids repeated Python calls for individual events. Events are processed in temporal order using the fusion target selection LUT. For an event batch, we pass the final temporal feature of each updated fusion target to the graph network once.

CUDA graph operations. We preallocate GPU buffers for nodes, edges, and intermediate features. CUDA operators append new graph connections without rebuilding the existing graph. The first graph block combines convolution, normalization, and residual updates in a single extension call, reducing dispatch overhead. It updates affected node features and reuses the cached features of other nodes. The first pooling stage also keeps its state on the GPU and updates affected cells. Layers 2–5 execute at query time using fixed spatial slots and masks for active nodes and neighbors. This avoids repeatedly constructing variable-sized graph tensors. Graph convolutions use precomputed spline-kernel weights indexed by spatial offsets.

CUDA Graph replay. The detection head uses fixed-size input buffers and masks for padded nodes, allowing its operations to be captured and replayed with CUDA Graphs. Replay reduces CPU kernel-launch overhead, while the masks prevent padded nodes from changing the output. Image features are cached and reused between event updates. These optimizations preserve the trained weights and node retention rule. We further optimize the official DAGr implementation using fused CUDA graph operators and CUDA Graph replay of the detection head, while preserving its graph

construction and model weights. Before these optimizations, it took 106.97 ms per 1 ms event batch in our measurements on an NVIDIA L40. All DAGr processing time results in this paper use the optimized implementation.

C.3 Effect of Node Retention

We evaluate CSEG under different node retention settings on DSEC-Det. Table S1 reports the resulting node retention and mAP.

Table S1: **Node retention and detection accuracy under different node retention settings.**

Setting (d_s, d_t, θ)	Retained (%)	mAP (%)
(2 px, 5 ms, 0.421000)	21.03	40.06
(1 px, 3.5 ms, 0.421000)	37.03	42.02
(1 px, 3.5 ms, 0.413739)	44.23	42.17
(1 px, 3.5 ms, 0.410256)	62.70	41.04
(1 px, 3.5 ms, 0.409384)	88.17	38.34

CSEG achieves its highest mAP at 44.23% retention and maintains 40.06% mAP when retaining only 21.03% of event nodes.

D Code Availability

We provide a runnable demo for review. The full code will be made publicly available upon acceptance.

References

Daniel Gehrig and Davide Scaramuzza. Low-latency automotive vision with event cameras. *Nature*, 629(8014):1034–1040, 2024. doi:10.1038/s41586-024-07409-w.