Supplemental Material for WACV2026 Submission

CoL^{2}A: C̄ōnvolution-free L̄ocal L̄inear Ātten̄tion for SpatioTemporal Event Processing

1. Efficient Associative Scan Implementation

Figure S1 provides our PyTorch [43] implementation for the stable, efficient temporal contraction of Z from Eq. (9) in the main paper. We adopt a prefix-sum–based approach inspired by Heinsen's algorithm [27], which reduces the naive $\mathcal{O}(T^2)$ complexity to approximately $\mathcal{O}(T \log T)$.

The direct implementation of Eq. (9) (`log_Z_contraction_complex`, Fig. S1-top) already provides a significant speedup over the naive $\mathcal{O}(T^2)$ algorithm. However, it uses logarithms for negative values, necessitating the computationally demanding complex value operation. By separately handling positive and negative parts and then combining them, we avoid complex-valued logarithms entirely as follows:

$$H_{+/-} = \exp\left(\Gamma^* + \log\left(Z^*_{+/-}\right)\right) \tag{10}$$

$$\Gamma^* = \sum_n^{cum} \Gamma_n, Z^*_{+/-} = \sum_n^{cum} \exp(\log Z_{+/-,n} - \Gamma^*) \tag{11}$$

$$H = H_+ - H_- \tag{12}$$

where, $Z_+ = \mathrm{ReLU}(Z)$ and $Z_- = \mathrm{ReLU}(-Z)$. This change yields approximately a 4× speedup in our experiments.

```python
def complex_log(x, eps=1e-6):
    real = x.abs().maximum(x.new_tensor(eps)).log()
    imag = (x < 0).to(x.dtype) * torch.pi
    return torch.complex(real, imag)
@torch.compile
def log_Z_contraction_complex(Gamma, Z, cum_eps=1e-6):
    Gamma_star = torch.cumsum(Gamma)
    Z_star = complex_log((Z).relu().add(cum_eps))- Gamma_star
    H = (
        (torch.logcumsumexp(Z_star ,dim=0).add(Gamma_star)).exp()
    )
    return H
```

```python
@torch.compile
def log_Z_contraction_sep(Gamma, Z, cum_eps=1e-6):
    Gamma_star = torch.cumsum(Gamma)
    Z_star_p = ( Z).relu().add(cum_eps).log()- Gamma_star
    Z_star_m = (-Z).relu().add(cum_eps).log()- Gamma_star
    H = (
        (torch.logcumsumexp(Z_star_p ,dim=0).add(Gamma_star)).exp()
        -
        (torch.logcumsumexp(Z_star_m ,dim=0).add(Gamma_star)).exp()
    ).real()
    return H
```

Figure S1. Pytorch implementation of efficient log-space associative scan. Implementation of Eq. (9) using complex log (`log_Z_contraction_complex`), and positive/negative separate implementation (`log_Z_contraction_sep`) of Eq. (10). With `@torch.compile`, the separate implementation (`log_Z_contraction_sep`) executes positive and negative parts in parallel, yielding substantial speedup over the one adopting complex operation.

2. Schematic illustration for deriving CoL^{2}A from convolutional PE

Fig. S2 illustrates the intuitive derivation of *CoL*^{2}*A* (Eq. (8)) from local positional embedding using convolution (Eq. (6), or equivalently local PE by expansion of Eq. (5)).

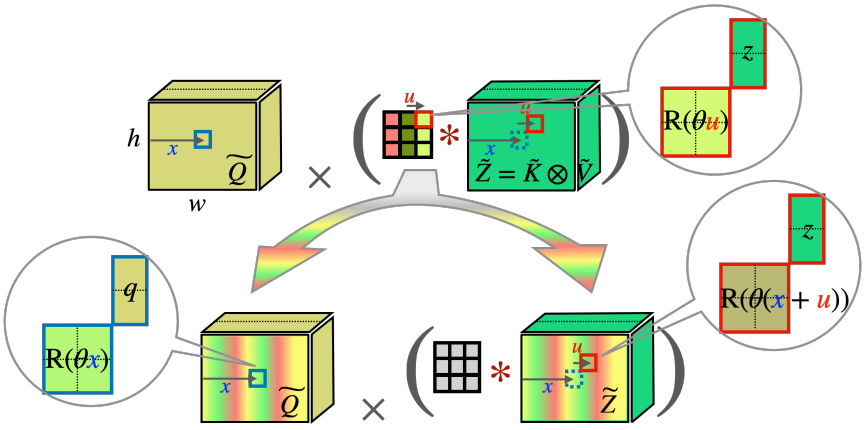

Figure S2. **Schematic derivation of *CoL*^{2}*A***. Idea is restrict the kernel to the special one (e.g., rotation using local coordinates u) and decompose the contribution as the rotation of Q, K matrices using global coordinates x (Sec. 4.3). This illustration only depicts the horizontal position for simplicity; however, actually, summation of the horizontal and vertical position are used.

3. Detailed experiment setup for ablation using ViT

Table S1 summarize the detailed setting for training the ViT network reported in Sec. 5 Tab. 7 and Tab. 8 in the main paper. On both experiments, we used ViT-Tiny [18], and LLA and *CoL*^{2}*A* used 7x7 size kernel for convolution and local averaging respectively. The settings for the classification are adopted from DeiT [64]. In the dense feature distillation task from DINOv3 [57], we used the pre-trained ConvNext-Base model [35][1] as a teacher (instead of ViT series) to better evaluate the capability of local modeling. We distill 14x14 features from the teacher model using mean squared error (MSE) loss.

Table S1. Settings for image classification and foundation model distillation on ImageNet1K [16].

Tasks	Classification	Feature Distillation
Epochs	300	100
Batch size	1024	1024
Optimizer	AdamW	AdamW
learning rate	$0.0005 \times \frac{\text{batchsize}}{512}$	$0.0005 \times \frac{\text{batchsize}}{512}$
Learning rate decay	cosine	cosine
Weight decay	0.05	0.05
Warmup epochs	5	5
Label smoothing ε	0.1	N/A
Stoch. Depth	0.1	0.1
Repeated Aug	✓	✓
Rand Augment	9/0.5	-
Mixup prob.	0.8	0.0
Cutmix prob.	1.0	0.0
Erasing prob.	0.25	0.0

[1] https://github.com/facebookresearch/dinov3

4. *CoL²A* block

Fig. S3 illustrates the *CoL²A* block. A single *CoL²A* layer includes a *CoL²A* module followed by layer normalization (LN) and a feedforward network (FFN), analogous to a standard transformer block. Stacking multiple blocks forms the *CoL²A* network. The cross-attention variant of *CoL²A* using a shared learnable query is adopted for the dense prediction head. GLA, LLA, and softmax attention used in the ablation experiments replace the *CoL²A* layer keeping the rest unchanged.

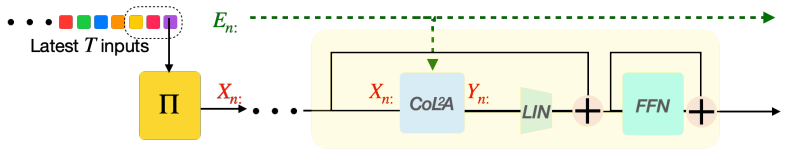

Figure S3. ***CoL²A* block**. Adopting a standard Transformer style architecture [18]. FFN: Feedforward network. Π: input embedding.

5. Scale equivariance

Fig. S4 shows a qualitative demonstration of scale equivalence of *CoL²A*. Because *CoL²A*'s local kernel is not discrete grid-based (like convolution), but in the form of a continuous function, its response to the scaled input would be equivariant by simply changing the area of local sum. Exploring the extension of *CoL²A* to have scale or other types of equivariance (e.g., rotational, deformation) would be an interesting research direction.

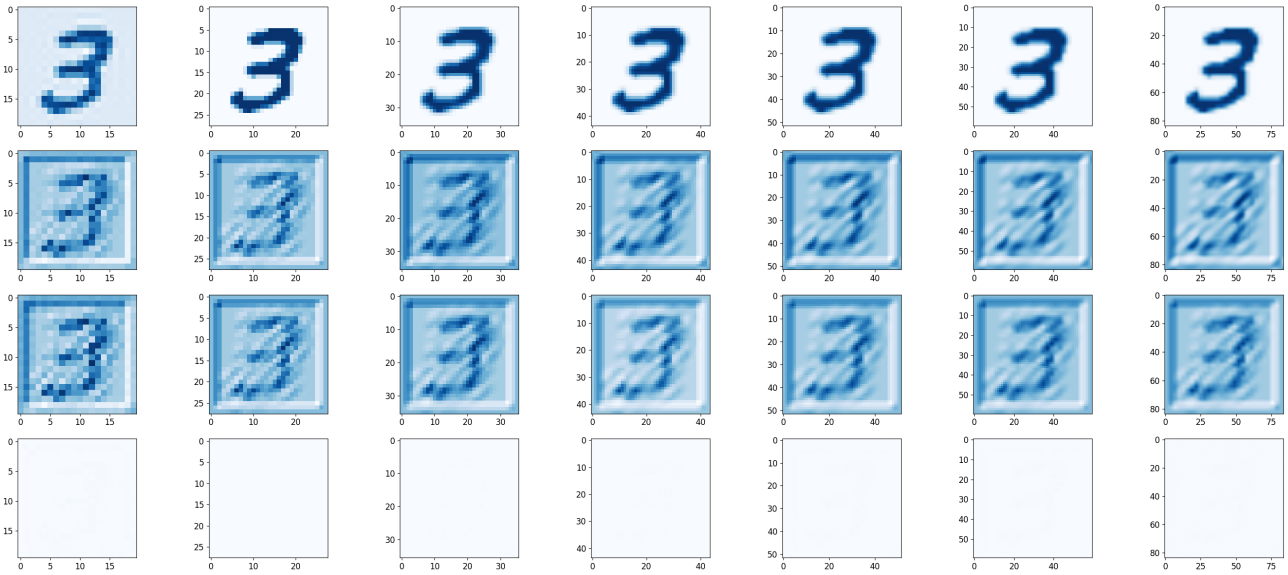

Figure S4. Scale equivariance demo of *CoL²A*. Top row: Input image $X_{s\times s}$ with varying size ($s \times s$: 18x18, 28x28(default), 34x34, 42x42, 58x58, 96x96). Second row: extracted feature from randomly initialized *CoL²A* layer f for each resized input, $f(X_{s\times s})$. Third row: Resized output feature extracted using default (28x28) size input, $\text{resize}(f(X_{28\times 28}), s \times s)$. Last row: Equivariance error, i.e., $f(X_{s\times s}) - \text{resize}(f(X_{28\times 28}), s \times s)$.

6. Comparison with Mamba-2 and Event-SSM

When kernel size κ of local sum in Eq. (8) spans the entire image (i.e., entire pixels share single memory), *CoL*^{2}*A* effectively reduces to Mamba-2 [14], with minor changes for sparse event data: i) *CoL*^{2}*A* uses a 2D extension of RoPE [28] for positional embedding, whereas Mamba-2 handles 1D sequences without PE; ii) *CoL*^{2}*A* does not have temporal convolutions immediately after Q, K, V projection. We define this model as GLA which can be regarded as an upgrade of Event-SSIM [52] replacing S5 [59] with the Mamba-2 [14] type linear attention Sec. 5.

As we saw on the ablation in the main paper, applying Mamba-2 style linear attention, i.e., ones using global memory, results in corrupted reconstructions on the event-to-video task (additional results are presented in Fig. S5, Tab. S3), failing to find valid keypoints on the keypoint detection task (additional results are presented in Tab. S2). We hypothesize that this could be mitigated by significantly enlarging memory, but the significant demand for memory (and computing) prohibits us from verifying this. We could not increase the memory dimension further ($d_f = d_{qk} = d_v = 96$) due to the limit of our H200 GPU with 140GB memory. These observations motivated us to develop *CoL*^{2}*A* Eq. (8), allowing efficient modeling of fine-grained spatiotemporal details through low dimensional local memory.

7. Other Input Representations or Event Filtering

In the main paper, we adopt the voxel representation from raw event using Eq. (1) to ensure a fair comparison with FireNet [11, 54]. By choosing a robust and sparse representation, we expect better accuracy while further reducing computation. Notably, our *CoL*^{2}*A* benefits from the sparsity of data; the sparser the input, the more computation reduces.

Several studies are offering the advanced filtering method for compressed event representations. For example, Inceptive Events [4] designates the first event in each triggered burst within a specified interval as the "inceptive" event and accumulates the subsequent events' polarities into it. Time-Surface [58], which builds a continuous memory surface of recent events, could also be used for compressing the event stream. In these cases, it is possible to utilize the precise event time (e.g., timestamp of incentive event instead of the discretized one with Δ_{in}) by incorporating it to Γ in Eq. (3).

8. Optimizing Kernel Size for Local Sum

In *CoL*^{2}*A* (Eq. (8) of the main paper), local sum is used to aggregate features within a local spatial window of size $\kappa \times \kappa$. Although this involves no multiplications, the complexity grows with larger κ. Moreover, choosing different values of κ for horizontal and vertical directions can introduce additional hyperparameters, potentially complicating tuning.

Fourier-Space Local Sum. To alleviate these concerns, one can perform a band-pass filter in the frequency domain, effectively learning a Gaussian mask parameter σ_x, σ_y during training:

$$H = \mathcal{F}^{-1}\Big(\mathcal{F}(Z) \odot M(\sigma_x, \sigma_y)\Big),$$

where $\mathcal{F}$ is the 2D FFT operator, $M(\sigma_x, \sigma_y)$ is a Gaussian mask in the Fourier domain parameterized by the standard deviation along horizontal and vertical axis σ_x, σ_y. It is initialized such that σ_x, σ_y is large enough to capture the entire spatial dimension (e.g., σ_x, σ_y). During training, the model can learn an appropriate local effective receptive field by adjusting $\sigma_x = w, \sigma_y = h$.

Post-training, one can either continue using this Fourier-based pooling or revert to a real-space kernel based on the learned $\sigma_x = w, \sigma_y = h$. One could expect faster computation using the Fourier-based approach when using a larger kernel.

9. Image Reconstruction Algorithm

In the main paper, we followed the recent approach for image reconstruction as presented in [20]. We estimate image gradients rather than raw intensity images, then apply Poisson integration for final reconstruction. This method has been shown to reduce the network size without noticeably harming performance.

We adopt the Frankot–Chellappa [19] integration algorithm for the Poisson Integration:

$$\mathcal{F}[s](f_x, f_y) = \frac{-if_x\,\mathcal{F}[s_x] \;-\; if_y\,\mathcal{F}[s_y]}{2\pi(f_x^2 + f_y^2)} \;+\; 0.5,$$

where s_x and s_y are the estimated horizontal and vertical gradients, $\mathcal{F}$ is the 2D FFT operator, and f_x, f_y are frequency coordinates.

Since the original implementation of FireNet [54] and ET-Net [71] directly estimates the intensity, we retrained them from scratch using the gradient prediction for comparisons.

10. Additional Results

Table S2 provides additional results on ATIS Corner dataset [38], extending Tab. 3 from the main text. Figure S5 shows further results on the High-Speed and HDR (HSHDR) dataset [48] , extending Fig. 5 in the main paper. Table S3 provides scene-wise results on the High Quality Frames (HQF) dataset [60], extending Tab. 4 from the main text.

Table S2. Corner tracking performance comparison on ATIS Corner dataset [38] (for δt ms interval). (Complete result of Tab. 3.)

	δt-Reprojection Error (pix) ↓					Track length (sec) ↑
Method	25	50	100	150	200	
eHarris [40]	2.57	3.46	4.58	5.37	6.06	0.74
eFast [40]	2.12	2.63	3.18	3.57	3.82	0.69
Arc [2]	3.80	5.31	7.22	8.48	9.49	0.91
SILC [38]	2.45	3.02	3.68	4.13	4.42	1.12
GLA/ GLA-L	NA	NA	NA	NA	NA	0.0
FireNet [10]	5.08	6.53	11.1	7.67	8.51	8.84
CoL²A	3.59	5.16	4.98	5.65	6.82	13.0

Table S3. Image reconstruction accuracy comparison on the HQF dataset [60]. (Complete results of Tab. 4. GLA-L adopts q_{kv}=48)

	MSE ↓					SSIM ↑					LPIPS ↓				
	ETNet	GLA	GLA-L	FireNet	*CoL²A*	ETNet	GLA	GLA-L	FireNet	*CoL²A*	ETNet	GLA	GLA-L	FireNet	*CoL²A*
boxes	0.078	0.120	0.120	0.096	0.102	0.40	0.28	0.28	0.36	0.35	0.35	0.38	0.38	0.36	0.36
desk fast	0.053	0.096	0.090	0.078	0.077	0.52	0.37	0.38	0.45	0.46	0.33	0.38	0.38	0.35	0.35
desk hand..	0.065	0.082	0.088	0.084	0.081	0.53	0.47	0.47	0.48	0.49	0.31	0.31	0.31	0.32	0.31
desk slow	0.047	0.085	0.094	0.091	0.086	0.59	0.44	0.44	0.49	0.48	0.30	0.35	0.35	0.33	0.34
poster pillar 1	0.062	0.071	0.071	0.065	0.066	0.39	0.30	0.30	0.35	0.35	0.34	0.35	0.35	0.35	0.35
poster pillar 2	0.053	0.053	0.052	0.050	0.051	0.46	0.41	0.42	0.44	0.44	0.31	0.30	0.30	0.32	0.31
reflective	0.075	0.091	0.093	0.078	0.082	0.40	0.30	0.30	0.36	0.36	0.34	0.36	0.36	0.35	0.36
slow hand	0.074	0.088	0.091	0.084	0.083	0.39	0.31	0.31	0.35	0.35	0.34	0.36	0.36	0.36	0.36
Mean	0.063	0.086	0.087	0.078	0.079	0.46	0.36	0.36	0.41	0.41	0.33	0.35	0.35	0.34	0.34

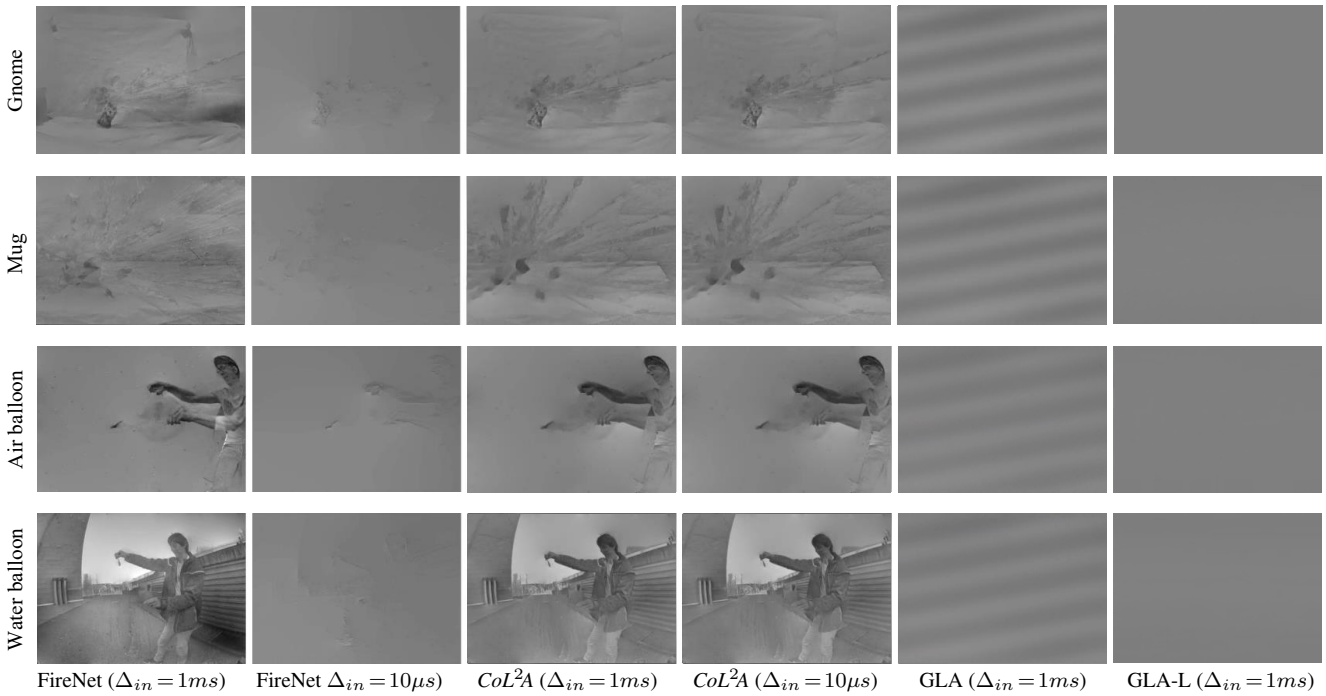

Figure S5. **Image Reconstruction** on HSHDR dataset [48]. Same configuration as in result on Fig. 5. GLA (Event-SSM [52] adopting Mamba-2 [70] for SSM) using the same memory dimension with *CoL²A* ($d_f = d_{qk} = d_v = 12$ with two heads). GLA-L use 4x more channel dimension. Even with increase channel, the model without locality failed to recover the meaningful image.