Moving Border Ownership for Event-based Motion Segmentation

Supplementary Material

8.1. Parameter-Efficient Adaptation with ConvLoRA Extended Formulations

Motivation. To adapt our pretrained streaming model from synthetic to real event data without overfitting, catastrophic forgetting, or large memory budgets, we employ low-rank adapters inserted into convolutional layers [1]. The base weights are frozen after training on synthetic data; only the lightweight adapters are trained, yielding strong domain transfer with a small fraction of parameters. We tested the domain transfer on the EVIMO2 dataset.

Formulation. Consider a convolution with frozen kernel $W_0 \in \mathbb{R}^{C_{\text{out}} \times C_{\text{in}} \times k \times k}$ and input x. ConvLoRA augments the layer with a low-rank update ΔW parameterized by two trainable matrices $B \in \mathbb{R}^{(C_{\text{out}} k^2) \times r}$ and $A \in \mathbb{R}^{r \times (C_{\text{in}} k^2)}$ of rank r, scaled by $s = \alpha / r$. The update is reshaped to match the kernel dimensions:

$$\Delta W := \text{reshape}(B\,A) \in \mathbb{R}^{C_{\text{out}} \times C_{\text{in}} \times k \times k},$$

$$y = (W_0 * x) + s\,(\Delta W * x),$$

where $*$ denotes convolution. In practice, W_0 is kept fixed; A and B are initialized such that the LoRA path is initially near-zero (e.g., B zeros), so the training starts from the pretrained behavior.

Parameter count. The additional trainable parameters per adapted conv are

$$\#\theta_{\text{LoRA}} = r\,(C_{\text{in}} k^2 + C_{\text{out}} k^2),$$

which totals 223,584 trainable adapter weights, while freezing the 16.8M parameters of the backbone. ConvLoRA accounts for 1.3% of the total parameter count. Thus, we can adapt several sensor-finetuning-specific blocks while keeping the total trainable parameters small.

Placement. We insert ConvLoRA adapters in task-specific convolutional modules (decoder and prediction heads) and lightweight projection layers around the temporal bottleneck (see Fig. 3). This concentrates adaptation capacity where domain shift is most pronounced, while leaving generic low-level encoders frozen.

Training. Only A, B (and optionally the bias in the prediction heads) are optimized; all other weights remain frozen. We use the same streaming losses as in Sec. 3.6. Typical hyperparameters are rank $r \in \{4, 8, 16\}$, scaling $\alpha \approx 2r$, and optional LoRA-path dropout $p \in [0, 0.1]$ for regularization. Group Normalization remains unchanged and is robust with batch size 1 for test-time inference.

Inference and merging. After finetuning, the low-rank update can be merged into the frozen kernel,

$$W_{\text{merged}} = W_0 + s\,\Delta W,$$

yielding a single convolution with *no* runtime overhead compared to the original model (the LoRA path can then be discarded). When further adaptation is desired, the merge is skipped and adapters remain active.

Finetune Performance. Please refer to section 6.4 for the validation of the finetune performance on the EVIMO2 dataset.

8.2. Ablation: Temporal Recurrence vs. Single-Frame

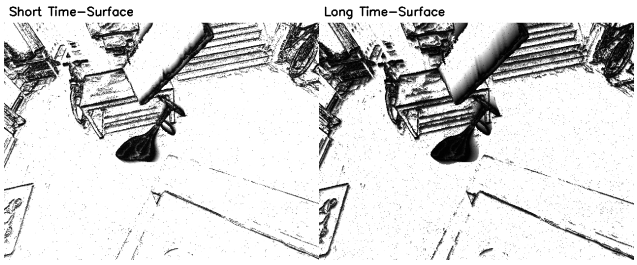

Figure 6. Example short and long duration time-surface used to train the single-frame method

Setup. To isolate the benefi of ConvLSTM, we compare our streaming approach against a "Single-Frame" baseline without recurrent memory. To compensate for the lack of temporal history, the baseline receives two time-surfaces ending at the same timestamp: a *Short* surface (10 ms) to provide sharp, blur-free border ownership cues, and a *Long* surface (30 ms) to capture sufficient motion context for independent object inference (visualized in Fig. 6).

Method	Input	FPS	mIoU
Single-Frame	Stacked (10ms + 30ms)	∼200	78.38
Ours (Temporal)	**Stream (10ms)**	**∼200**	**81.55**

Table 7. Temporal Recurrence vs. Single-Frame Inputs evaluated Zero-Shot on EVIMO2.

Results. As shown in Table 7, the Single-Frame model achieves 78.38 mIoU on EVIMO2 (zero-shot). While the combined short/long inputs capture significant motion signals, the proposed Temporal model outperforms it with 81.55 mIoU. This confirms that explicit recurrent-state modeling handles dynamic occlusions more effectively than simply extending the input duration.

Runtime. Both architectures achieve a similar throughput of ∼200 FPS on one Nvidia RTX2080Ti; the Single-Frame model avoids ConvLSTM gate computations but incurs overhead from processing the additional input channel.

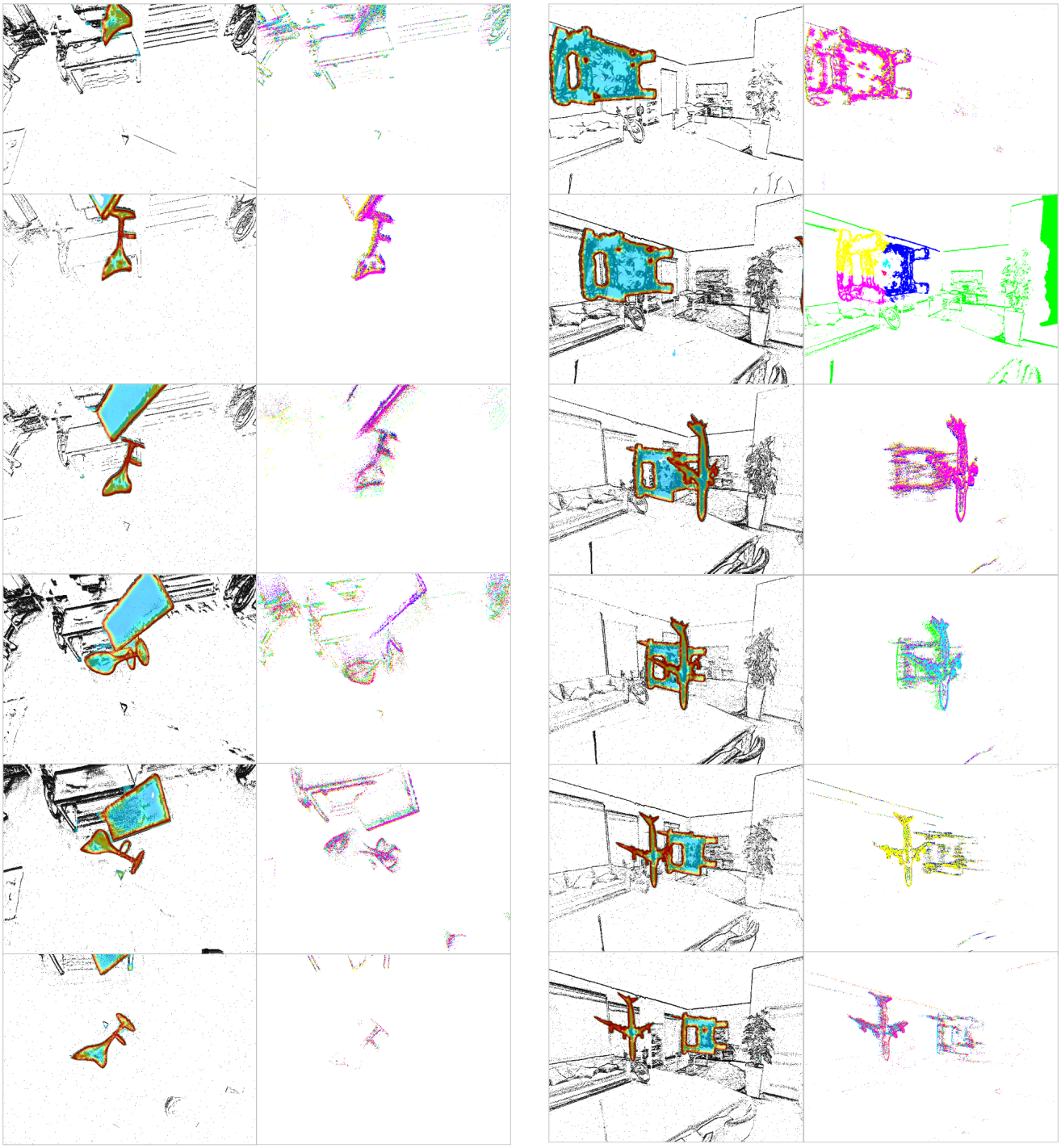

Figure 7. Visual comparison with EMSGC. For our method, red boundaries indicate predicted border ownership, and the yellow-to-blue gradient visualizes the ownership field showing which side owns the border. For EMSGC, colors signify different objects (if optimized correctly, only moving objects should be visualized).

8.3. Qualitative Comparison with Geometric Clustering

We further analyze the robustness of our method by comparing it against EMSGC [41], a leading geometric motion-clustering baseline. Figure 7 highlights two challenging scenarios where model-based assumptions often fail.

Videos of these visualizations are also included in the supplementary material.

On the left, we focus on a scenario where two objects move with similar motion direction while the camera also moves in the same direction. This configuration is often a degenerate case for motion-clustering methods, which struggle to distinguish the two objects apart or separate them from the background. Our method is the only one able to segment out both objects consistently while distinguishing clear borders for each; in contrast, EMSGC clusters them into the same motion cluster, signified by the same color group, failing to separate the instances; EMSGC also fails in removing the static background in some frames.

On the right, we present a challenging occlusion case where two objects interact. Our method reliably separates the foreground moving object from the background occluded moving object, maintaining distinct ownership boundaries. EMSGC, however, merges the interacting objects into a single cluster (visualized by the uniform color), and failing to remove the static background in some frames, demonstrating the limitation of pure motion grouping in resolving complex occlusions.

Runtime. Our method runs twice as fast as real time at 200 FPS on one Nvidia RTX 2080 Ti. The EMSGC algorithm fails to converge at full-resolution events. On a 3x downsampled event, it uses approximately 1 minute to process each frame.