ARES: Unifying Asymmetric RGB-Event Stereo for Probabilistic Scene Flow Estimation

Supplementary Material

1. Network Architecture Details

We extract features at $\times 1/8$ and $\times 1/16$ spatial resolutions for the RGB and event modalities, respectively, while the bridging representation operates at $\times 1/8$ resolution. This configuration achieves a balanced trade-off between spatial precision and temporal expressiveness. The higher spatial resolution of RGB features preserves fine image structure and texture cues that are essential for accurate stereo matching, whereas the deeper event encoder at $\times 1/16$ resolution captures temporally aggregated motion information with reduced spatial redundancy. The bridging features at $\times 1/8$ resolution serve as a common alignment space, maintaining compatibility with the RGB encoder while benefiting from the temporal awareness of event representations. Processing MCA features at $\times 1/8$ resolution also offers a favorable efficiency–accuracy compromise: it lowers the quadratic memory complexity of attention while retaining semantically rich tokens that effectively summarize the geometry and dynamics of the scene.

1.1. Transformer Configuration

We use two transformer blocks, each with interleaved attentions as described in Eqn. 13 of the main paper. Each attention layer comprises one head with 128 hidden dimensions, with positional encoding provided by Axial RoPE. To reduce memory consumption, we adopt memory-efficient attention enabling training on 640×480 inputs without gradient checkpointing.

1.2. Encoder Architectures

Bridge Encoder. Given a bridging image (temporal integral or temporal difference) $x \in \mathbb{R}^{B\times 3\times H\times W}$, we initialize and train an EdgeNeXt model for feature extraction:

$$\begin{aligned} x_{\text{stem}} &= \text{Stem}(x), \\ x_4 &= \text{Stage}_1(x_{\text{stem}}), && \in \mathbb{R}^{B\times 48\times \frac{H}{4}\times \frac{W}{4}}, \\ x_8 &= \text{Stage}_2(x_4), && \in \mathbb{R}^{B\times 96\times \frac{H}{8}\times \frac{W}{8}}, \\ x_{16} &= \text{Stage}_3(x_8), && \in \mathbb{R}^{B\times 160\times \frac{H}{16}\times \frac{W}{16}}, \\ x_{32} &= \text{Stage}_4(x_{16}) && \in \mathbb{R}^{B\times 304\times \frac{H}{32}\times \frac{W}{32}} \end{aligned} \tag{1}$$

We then perform top–down fusion using Conv2x_IN blocks:

$$\begin{aligned} \hat{x}_{16} &= \text{Conv2x\_IN}^{32\to 16}(x_{32},\, x_{16}), \\ \hat{x}_8 &= \text{Conv2x\_IN}^{16\to 8}(\hat{x}_{16},\, x_8), \end{aligned} \tag{2}$$

where each $\text{Conv2x\_IN}^{a\to b}$ upsamples the first input feature by a factor of 2, concatenates it with the second, and applies instance-normalized convolutions. Finally, we refine $\hat{x}_8$ and project to 128 channels:

$$\begin{aligned} u_8^{(1)} &= \text{ResBlock}_{\text{IN}}(\hat{x}_8), \\ u_8^{(2)} &= \text{ResBlock}_{\text{IN}}(u_8^{(1)}), \\ f_{1/8} &= \text{BasicConv}_{3\times 3}^{192\to 128}(u_8^{(2)}) \in \mathbb{R}^{B\times 128\times \frac{H}{8}\times \frac{W}{8}}. \end{aligned} \tag{3}$$

The bridge encoder produces output features $f_{1/8} \in \mathbb{R}^{B\times 128\times \frac{H}{8}\times \frac{W}{8}}$ from the bridging images.

RGB Encoder. Given an RGB image $x \in \mathbb{R}^{B\times 3\times H\times W}$, a frozen DINOv3 ConvNeXt backbone produces hidden states

$$\text{DINO}(x) \Rightarrow \{h_0, h_1, h_2, h_3, h_4\}, \tag{4}$$

with spatial feature maps

$$\begin{aligned} h_1 &\in \mathbb{R}^{B\times 96\times \frac{H}{4}\times \frac{W}{4}}, \\ h_2 &\in \mathbb{R}^{B\times 192\times \frac{H}{8}\times \frac{W}{8}}, \\ h_3 &\in \mathbb{R}^{B\times 384\times \frac{H}{16}\times \frac{W}{16}}, \\ h_4 &\in \mathbb{R}^{B\times 768\times \frac{H}{32}\times \frac{W}{32}}. \end{aligned} \tag{5}$$

We use the level 3 features and take

$$f = h_3 \in \mathbb{R}^{B\times 384\times \frac{H}{16}\times \frac{W}{16}}.$$

We then apply a lightweight residual adapter:

$$\begin{aligned} x_{\text{norm}} &= \text{GN}(f), \\ a &= \text{Conv}_{1\times 1}^{384\to d_b}(x_{\text{norm}}), \\ a &= \text{GN}(a), \quad a = \text{GELU}(a), \\ a &= \text{Dropout2d}(a), \\ a &= \text{Conv}_{1\times 1}^{d_b\to 384}(a), \\ \tilde{f} &= x_{\text{norm}} + \alpha\, a, \end{aligned} \tag{6}$$

with bottleneck width $d_b = 64$ and learnable scale α.

Finally, we project to the desired output dimension $d_{\text{out}} = 128$:

$$\begin{aligned} z &= \text{GN}(\tilde{f}), \\ z &= \text{Conv}_{1\times 1}^{384\to 128}(z), \end{aligned} \tag{7}$$

and use $z \in \mathbb{R}^{B\times 128\times \frac{H}{16}\times \frac{W}{16}}$ as the final DINOv3 feature map.

Event Encoder. Each event stream $\mathcal{E} = \{(x_i, y_i, t_i, p_i)\}$ is voxelized over B temporal bins:

$$V(x, y, b) = \sum_i p_i \cdot \max\left(0, 1 - \frac{|b - B(t_i)|}{B}\right), \quad (8)$$

where $p_i \in \{-1, +1\}$ is the event polarity. The resulting voxel grid is normalized by the total number of events in the window to maintain scale invariance.

Given voxelized events $V \in \mathbb{R}^{B\times 15\times H\times W}$, the encoder applies:

$$\begin{aligned} c_0^{\text{conv}} &= \text{Conv}_{7\times 7, s=2}(V), \\ c_0^{\text{in}} &= \text{IN}\big(c_0^{\text{conv}}\big), \\ c_0 &= \text{ReLU}\big(c_0^{\text{in}}\big), \\ \text{net}_0 &= \text{ConvGRU}_0\big(\text{net}_0^{\text{prev}},\, c_0\big), \end{aligned} \quad (9)$$

with $\text{net}_0^{\text{prev}} = c_0$ at the first time step.

For layers $\ell = 1, 2, 3, 4$:

$$\begin{aligned} c_\ell^{\text{in}} &= \big[c_{\ell-1},\, \text{net}_{\ell-1}\big], \\ c_\ell &= \text{ResBlock}_\ell\big(c_\ell^{\text{in}}\big), \\ \text{net}_\ell &= \text{ConvGRU}_\ell\big(\text{net}_\ell^{\text{prev}},\, c_\ell\big). \end{aligned} \quad (10)$$

where the spatial resolutions are:

$$\begin{aligned} c_1, \text{net}_1 &: \ \tfrac{H}{2} \times \tfrac{W}{2}, \\ c_2, \text{net}_2 &: \ \tfrac{H}{4} \times \tfrac{W}{4}, \\ c_3, \text{net}_3 &: \ \tfrac{H}{8} \times \tfrac{W}{8}, \\ c_4, \text{net}_4 &: \ \tfrac{H}{16} \times \tfrac{W}{16}. \end{aligned} \quad (11)$$

Multi-scale outputs are:

$$\begin{aligned} f_{1/2} &= \text{Conv}^{1\times 1}_{32\to 128}(\text{net}_1), \\ f_{1/4} &= \text{Conv}^{1\times 1}_{48\to 128}(\text{net}_2), \\ f_{1/8} &= \text{Conv}^{1\times 1}_{64\to 128}(\text{net}_3), \\ f_{1/16} &= \text{Conv}^{1\times 1}_{96\to 128}(\text{net}_4), \end{aligned} \quad (12)$$

and the encoder returns $f_{1/8}$ as the event context features.

TDPF CNN. Given a concatenated feature tensor:

$$X \in \mathbb{R}^{B\times(3+4C)\times H\times W}, \quad C = 128, \quad (13)$$

the TDPF CNN applies a stack of 3×3 convolutions with GroupNorm and GELU activations:

$$\begin{aligned} X_1 &= \text{GELU}(\text{GN}_8(\text{Conv}^{3+4C\to 256}_{3\times 3}(X))), \\ X_2 &= \text{GELU}(\text{GN}_8(\text{Conv}^{256\to 256}_{3\times 3}(X_1))), \\ X_3 &= \text{GELU}(\text{GN}_8(\text{Conv}^{256\to 128}_{3\times 3}(X_2))), \\ X_4 &= \text{GELU}(\text{GN}_8(\text{Conv}^{128\to 128}_{3\times 3}(X_3))), \\ X_5 &= \text{GELU}(\text{GN}_8(\text{Conv}^{128\to 64}_{3\times 3}(X_4))), \\ \Delta d_{\text{res}} &= \text{Conv}^{64\to 1}_{3\times 3}(X_5), \end{aligned} \quad (14)$$

where GN_8 denotes GroupNorm with 8 groups. The output is a single-channel residual disparity map $\Delta d_{\text{res}} \in \mathbb{R}^{B\times 1\times H\times W}$.

1.3. Posterior Interpolation

Disparity correlation volumes $C_{t_0}^L$ and $C_{t_1}^L$ are normalized with a softmax function along the disparity axis:

$$\bar{C}(d) = \frac{\exp(C(d))}{\sum_{d'} \exp(C(d'))}. \quad (15)$$

To ensure resolution invariance, each disparity posterior is linearly interpolated to a fixed 128 disparity bins, allowing TDPF to process both patch-level and full-frame inputs seamlessly.

1.4. Disparity–Flow Inference Pipeline

Following feature fusion in MCA, ARES performs disparity and optical flow estimation through two parallel yet structurally similar pipelines, both leveraging correlation volumes for dense correspondence reasoning and iterative refinement for accuracy.

Correlation Volume Construction. For disparity estimation, a left–right correlation volume $C_{\text{disp}} \in \mathbb{R}^{H\times W\times D}$ is computed between MCA features from the left and right views at the same timestamp t:

$$C_{\text{disp}}(x, y, d) = \langle F_t^L,\, F_t^R \rangle, \quad (16)$$

where $\langle \cdot, \cdot \rangle$ denotes the dot product along the channel dimension, and d indexes disparity hypotheses. Similarly, optical flow correlation is computed between temporally adjacent view-specific features:

$$C_{\text{flow}}(x, y, \Delta x, \Delta y) = \langle F_{t_0}^v,\, F_{t_1}^v \rangle. \quad (17)$$

Coarse Estimation via Softmax Averaging. Each correlation volume is normalized with a softmax function along the displacement dimension to obtain a probability distribution over correspondence hypotheses:

$$P(d) = \frac{\exp(C_{\text{disp}}(d))}{\sum_{d'} \exp(C_{\text{disp}}(d'))}, \quad (18)$$

$$P(\Delta x, \Delta y) = \frac{\exp(C_{\text{flow}}(\Delta x, \Delta y))}{\sum_{\Delta x', \Delta y'} \exp(C_{\text{flow}}(\Delta x', \Delta y'))}. \quad (19)$$

The coarse disparity and flow estimates are then computed as the expectation of these distributions:

$$\hat{D}_{\text{coarse}} = \sum_d d \cdot P(d), \quad (20)$$

$$(\hat{u}^L_{\text{coarse}}, \hat{v}_{\text{coarse}}) = \sum_{\Delta x, \Delta y} (\Delta x, \Delta y) \cdot P(\Delta x, \Delta y). \quad (21)$$

Iterative Refinement. Both disparity and flow predictions are refined using a RAFT-style iterative lookup and update process. At each iteration k, the coarse prediction is used to sample correlation features via bilinear interpolation from the correlation volume:

$$f_k = \mathcal{S}(C, \hat{p}_{k-1}), \tag{22}$$

where $\hat{p}_{k-1}$ represents the current disparity or flow estimate, and $\mathcal{S}$ denotes differentiable sampling. The sampled correlation features, together with the MCA features and the current estimate, are passed into a refinement network Φ_{ref} to predict a residual update:

$$\Delta\hat{p}_k = \Phi_{\text{ref}}\big(f_k, F_{\text{MCA}}, \hat{p}_{k-1}\big), \tag{23}$$

$$\hat{p}_k = \hat{p}_{k-1} + \Delta\hat{p}_k. \tag{24}$$

This refinement is repeated for three iterations ($k = 1, 2, 3$), progressively improving correspondence precision and geometric smoothness.

2. Training Protocol

2.1. Split Optimizers and Learning Schedule

To stabilize multi-task learning and reduce memory usage, we use separate optimizers for the MCA and TDPF components. The MCA encoder, responsible for cross-modal feature fusion and correspondence estimation, is trained independently from the TDPF network, which focuses on disparity evolution and scene flow reconstruction. This separation prevents conflicting gradient updates between spatial–temporal fusion and probabilistic refinement while allowing more efficient memory management.

MCA is optimized with AdamW using a cosine annealing schedule with linear warm-up to encourage stable convergence of transformer layers during early training. TDPF, in contrast, is trained with a constant learning rate, providing consistent updates for precise disparity-change refinement once flow and disparity predictions have stabilized. This decoupled optimization strategy ensures steady convergence across modules and leads to more reliable 3D motion estimation.

Loss weightings. Table 2 summarizes the relative weighting of the supervised and self-supervised loss terms used during training. These values balance photometric regularization with ground-truth flow and disparity supervision to ensure stable multi-task optimization.

2.2. Data Augmentation

We apply random scaling (0.8–1.2×), cropping, and vertical flipping to expand the diversity of training samples for improved generalization. All inputs (disparity, optical flow, bridging images, event voxel grids, RGB images) are spatially transformed identically to preserve cross-modal alignment.

Table 1. Loss weightings used in ARES training.

Loss Term	Symbol	Weight
Optical flow loss	$\mathcal{L}f$	1.0
Disparity loss	$\mathcal{L}d$	1.0
Photometric loss	$\mathcal{L}p$	1e-5
Disparity warping loss	$\mathcal{L}w$	1.0

2.3. Optical Flow Training

Since right-view flow supervision is unavailable, both left and right channels use the same MCA module with the same flow refinement model to enforce cross-view consistency. Furthermore, we train the flow pipeline on both left-view temporal difference and temporal integral bridging images, allowing the flow model to be able to generalize to the temporal difference images on the right view without requiring right-view supervision.

2.4. Two-Stage Training

On DSEC, training proceeds in two stages: **Stage 1 (Patch-level)** trains on 256×256 crops for local correspondence learning. **Stage 2 (Global)** fine-tunes on full-resolution frames (640×480) to improve global geometric consistency and long-range motion reasoning. For MVSEC, we train in a single stage at global scale.

Training schedule. We train for 400k iterations (per stage) with a batch size of 4 on one RTX A6000 GPU (48 GB). We train in mixed-precision (FP16) to improve memory efficiency.

3. Compute Cost

Table 2. Model size and Inference Time.

Method	Params (M)	Time (s)
ARES (Ours)	52	0.81
RAFT-3D	45	0.36
EMatch	2.25	0.13

We report parameter counts and inference times in Tab. 3. Although incurring slightly heavier computation, our ARES remains practical and improves scene flow accuracy.