DeDelayed: Deleting Remote Inference Delay via On-Device Correction

Supplementary Material

8.1. Code

The fused "local + remote predictive" model is defined as:

```
class FusedModel(nn.Module):
    def __init__(self):
        super().__init__()
        self.local_model = LocalModel()
        self.remote_model = RemoteModel()

    def forward(self, x_local, x_remote, delay):
        # Ensure remote z size matches local h size:
        H_local, W_local = x_local.shape[-2:]
        z_size = (H_local // 8, W_local // 8)
        # Receive remote features:
        z = self.remote_model(x_remote, delay, z_size, head="features")
        # Local model with optional fusion:
        y = self.local_model(x_local, z)
        return y


class RemoteModel(nn.Module):
    def __init__(self, name="efficientvit-seg-l1-cityscapes"):
        super().__init__()
        self.image_model = create_efficientvit_seg_model(name)
        self.video_model_backbone = nn.Sequential(*[
            VitBlock3D(in_channels=256, head_dim=32, expand_ratio=4)
            for _ in range(12)
        ])
        self.delay_embedding = DelayEmbedding(output_shape=(256, 1, 1, 1))
        self.seg_head = RemotePredictiveHead(seg_classes=19)
        self.mlp_pre_pool = PrepoolBlock()
        self.mlp_post_pool = PostpoolBlock()

    def forward(self, x_remote, delay, z_size, head):
        # Apply 2D image model backbone to each frame individually:
        z = x_remote
        z = einops.rearrange(z, "b c f h w -> (b f) c h w", f=4)
        z = self.image_model.backbone(z)
        z = einops.rearrange(z, "(b f) c h w -> b c f h w", f=4)
        # Add delay embedding element-wise, then run 3D ViT:
        z = z + self.delay_embedding(delay)
        z = self.video_model_backbone(z)
        # Collapse temporal axis into channel axis, and apply 2D head:
        z = einops.rearrange(z, "b c f h w -> b (c f) h w", f=4)
        if head == "logits":  # Send labels (no fusion).
            z = self.seg_head(z)
        elif head == "features":  # Send features (for fusion).
            z = self.mlp_pre_pool(z)
            z = F.adaptive_avg_pool2d(z, z_size)
            z = self.mlp_post_pool(z)
        return z


class LocalModel(nn.Module):  # Any 2D image segmentation model.
    def forward(self, x_local, z_remote=0):
        # Local feature extraction:
        h = self.T1(x_local)
        # Local and remote fusion:
        h = h + z_remote
        # Remaining local model:
        y = h
        h = self.T2(h)
        y = y + self.P2(h)
        h = self.T3(h)
        y = y + self.P3(h)
        y = self.seg_head(y)
        return y
```

During training and evaluation, the models are fed frames and target labels offset by the appropriate delay. Otherwise, training minimizes per-pixel cross-entropy loss and evaluation uses mIoU as usual.

8.2. Additional architecture details

Delay embedding. We use a two-layer MLP which maps $\mathbb{R} \to \mathbb{R}^C$ via layers of sizes 1024 and 256.

ViT3D. This consists of 12 blocks with 256 input/output channels, where each block contains a 3D attention followed by $3 \times 3 \times 3$ MBConv3D.

8.3. Effect of delay jitter

We evaluate how our model performs under delay jitter, i.e., when the delay varies over time. Our training loss targets accuracy for a fixed, tunable delay input—we do not explicitly train it to be jitter-resilient. Nonetheless, temporal structure in the data helps maintain accuracy even when the delay input differs from the observed delay. Fig. 8 characterizes this, showing performance across observed delays when the model is fed a possibly incorrect delay as input.

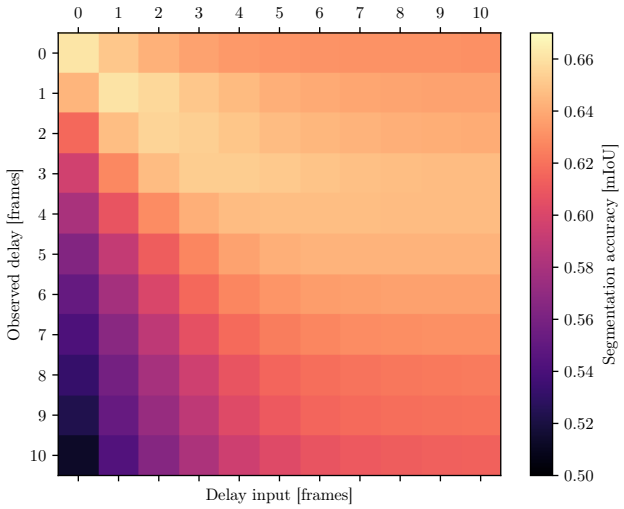

Figure 8. Segmentation accuracy (mIoU) over observed delay and model delay input.

Although the model was not explicitly trained for mismatched delays or delays beyond 5 frames, it still performs well at these out-of-distribution delays. Unsurprisingly, accuracy peaks when the model's delay input matches the observed delay. Notably, the accuracy drop is smaller when the delay input exceeds the observed delay than the reverse. Thus, when jitter is high, it is safer to use a larger delay input than expected, since underestimating the delay tends to make the remote features overconfident about localization.

This matrix can be precomputed during evaluation. As noted in Sec. 7, at runtime the device may consult this matrix to choose among the received remote feature tensors $\{z_1, z_2, \ldots\}$ by looking up the accuracy for each tensor's delay pair (D_o, D_i) and selecting the best. After a dramatic scene change or excessive staleness, the device may omit remote features from the local model input and run the local model alone, yielding its baseline performance.

8.4. Local input resolution

We evaluate performance across local input resolutions by finetuning the fused model (trained at 480 px) for 10 additional epochs at 224, 320, and 480 px. The results are shown in Fig. 9. Our remote-assisted local model operates at far lower resolutions (e.g., 224 px) while achieving better accuracy under round-trip latency than other solutions.

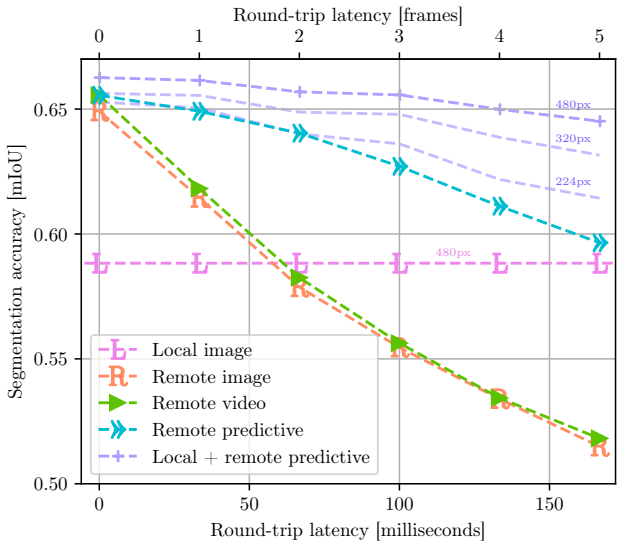

Figure 9. Segmentation accuracy (mIoU) versus round-trip latency (milliseconds or frames). Further finetuned and evaluated on various local input resolutions.

8.5. Delay-input ablation

We ablate the delay input by removing the delay conditioning from the remote model. This causes an average drop of 0.015 mIoU for local 224 px, showing that explicit delay inputs are beneficial.

Table 4. Delay-input ablation.

Experiment	Remote delay (frames) 0	1	2	3	4	5	Local
BDD 224px	0.652	0.650	0.640	0.635	0.621	0.613	—
BDD 224px (– delay)	0.622	0.628	0.627	0.625	0.615	0.603	—

8.6. Robustness to uplink compression

We evaluate performance under x264/x265 compression (veryfast, zerolatency, bframes=0) to assess robustness to realistic streaming configurations.

Table 5. Robustness to uplink compression.

Experiment	Remote delay (frames) 0	1	2	3	4	5	Local
BDD H.265 5 Mbps	0.658	0.657	0.652	0.648	0.642	0.635	—
BDD H.264 5 Mbps	0.657	0.656	0.651	0.647	0.640	0.634	—
BDD H.265 3 Mbps	0.655	0.654	0.649	0.644	0.640	0.635	—
BDD H.264 3 Mbps	0.652	0.652	0.647	0.644	0.637	0.632	—
BDD H.265 1 Mbps	0.631	0.630	0.627	0.623	0.619	0.615	—
BDD H.264 1 Mbps	0.612	0.610	0.608	0.605	0.603	0.599	—

8.7. Compute and runtime breakdown

Table 6. Compute and runtime breakdown by component.

Component	Params	MACs	GPU (ms)	CPU (ms)
Remote ViT2D	42.206M	199.683G	2.01	704.57
Delay embedding	0.264M	0.0005G	0.04	1.26
Remote ViT3D+head	11.964M	551.665G	24.38	34475.63
Local ViT2D+head	8.101M	1.455G	0.67	9.34
Additive fusion	0	0	0.04	0.01

GPU: NVIDIA H100 fp16 PyTorch compiled | CPU: Apple M3 Pro fp32 PyTorch 6 cores | Local: 224px.

8.8. Additional dataset evaluation

We train DeDelayed models on 35k clips from the first 229 sequences in the Nymeria dataset [28] and evaluate on 2k clips from 25 disjoint sequences with unseen environments and actions, using the same semantic segmentation task and ADE20K taxonomy.

Table 7. Generalizability to new environments and actions.

Experiment	Remote delay (frames) 0	1	2	3	4	5	Local
Nymeria [28]	0.2450	0.2451	0.2448	0.2446	0.2445	0.2443	0.2034

8.9. Occlusion

Fig. 10 shows a qualitative example of a newly unoccluded object missed by all methods except DeDelayed.

Remote (conventional) Local-only Dedelayed

Figure 10. Due to 167 ms RTT, the remote model has not yet seen any frames containing the recently unoccluded cyclist, so methods whose final inference is not local surely fail to detect it.

8.10. Sample validation pseudolabels

Fig. 11 visualizes 20 randomly selected samples from the validation set, with labels colored according to the conventional Cityscapes palette in Fig. 12. The ego vehicle's hood

is unlabeled, which is appropriate. Core classes are generally accurate and present with low error: road, sky, vegetation, building, pole, person, car, truck, bus. Most distant traffic lights and traffic signs are correctly identified. In (row 2, column 1), the area beneath the fence is labeled sidewalk, which is debatable; a nearby non-traffic sign is left unlabeled, which is reasonable. Some uncertain areas show speckles and jagged boundaries, often near partial occlusions. Overall, the pseudolabels are close to human-annotated quality, with few exceptions.

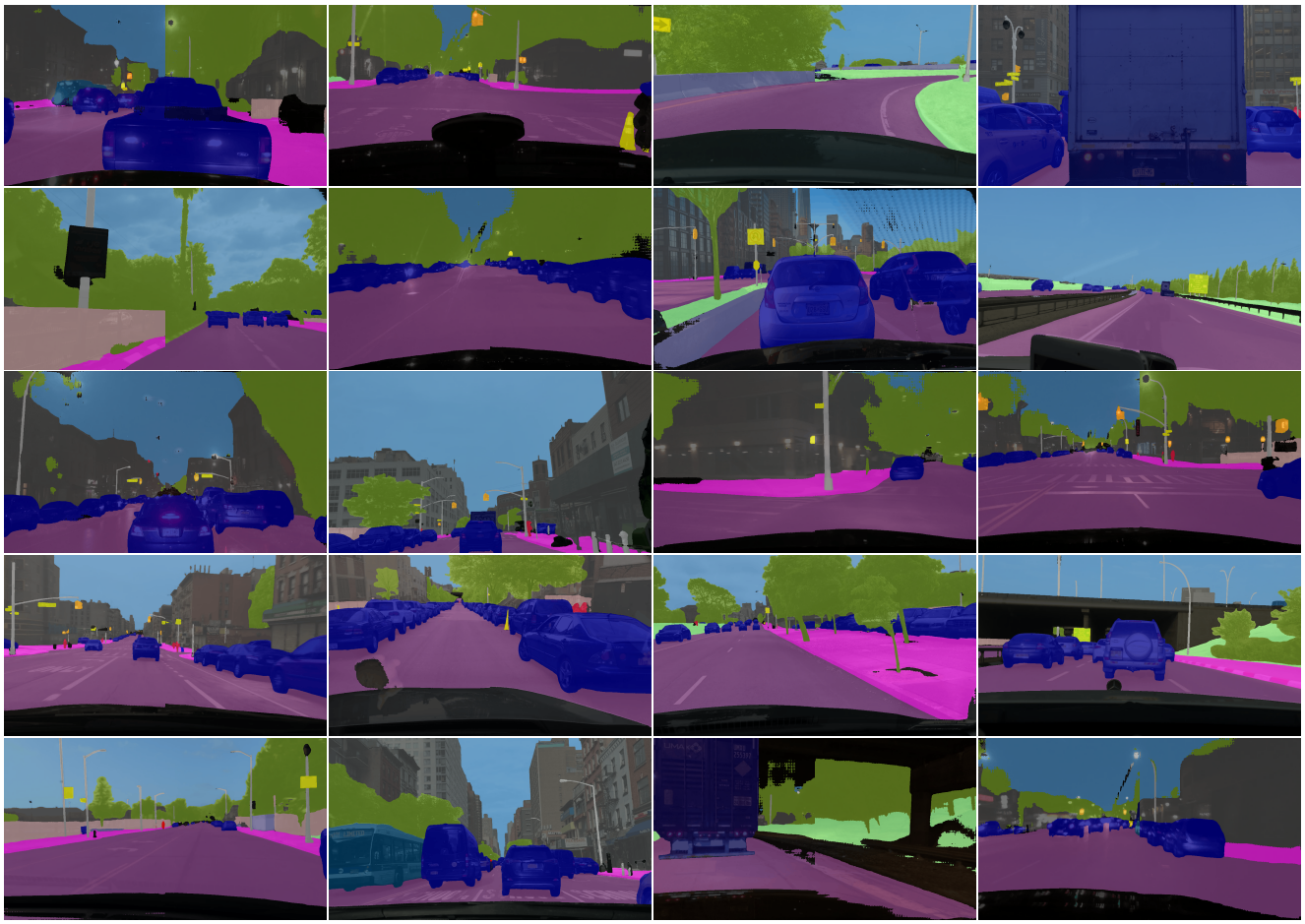

Figure 11. Randomly sampled validation pseudolabels.

unlabeled	wall	traffic sign	person	bus
road	fence	vegetation	rider	train
sidewalk	pole	terrain	car	motorcycle
building	traffic light	sky	truck	bicycle

Figure 12. Cityscapes class palette.