Supplementary Material:
Joint Forecasting of Panoptic Segmentations with Difference Attention

This appendix is structured as follows: Appendix A details the background prediction approach which we use to obtain preliminary background class predictions. Appendix B provides specific model architectural details for the forecasting transformer encoder and decoder. Appendix C explains in detail the agent-aware attention approach we use which allows for identity information to be encoded in the model. Appendix D describes the specific losses computed during training of the foreground forecasting model. Appendix E presents the model architecture used by the depth completion model introduced in Sec. 3.3. Appendix F describes additional details of implementation and model training. Appendix G contains additional information about the AIODrive dataset and experiments. Appendix H contains instance segmentation forecasting experimental results for Cityscapes. Appendix I contains semantic segmentation forecasting experimental results for Cityscapes. Appendix J presents additional model visualizations on Cityscapes for both the short- and mid-term settings. Appendix K contains the full per-class breakdown of the panoptic segmentation metrics presented in Tab. 1. Appendix L describes the major code libraries used to implement our model. Finally, Appendix M discusses potential negative societal impacts that could arise from the implementation of this work in practice.

A. Background Model

In this work, we utilize the background semantic prediction model introduced by Graber *et al*. [14]. This approach lifts background semantics into a 3D point cloud using the estimated input depth, transforms the point cloud based on camera movement, projects to the image plane, and refines the projected semantics using a semantic segmentation model. Formally, this model estimates the semantics of background object classes for unseen future frame $T+F$ as

$$\widehat{m}^B_{T+F} = \text{BGRef}(\{\text{proj}(m_t, d_t, K, H_t, u_t)\}_{1:T}), \tag{16}$$

where K represents camera intrinsic parameters, H_t is the 6-dof camera transform from input frame t to target frame $T+F$, m_t is the semantic segmentation for frame t which is obtained from a pre-trained model, d_t is the input depth map at time t, and u_t denotes the coordinates of all of the pixels in m_t which correspond to background semantic classes. *Proj* refers to the step which creates the sparse reprojected semantic map for frame $T+F$ given inputs for frame t, and *BGRef* refers to the background refinement model which produces a complete background prediction from the output of *Proj*.

The first step of the background model is to produce reprojected semantic point clouds $(\tilde{m}^B_t, \tilde{d}^B_t)$ which are processed by *BGRef*. These are obtained for each time $t \in \{1, \ldots, T\}$ by applying *Proj* to the corresponding input frame I_t. Given per-pixel semantic prediction m_t and depth map d_t, *Proj* back-projects, transforms, and reprojects the pixels from input frame t to target frame $T+F$. This process is summarized as

$$\begin{bmatrix} x_t \\ y_t \\ z_t \end{bmatrix} = H_t \begin{bmatrix} K^{-1} \begin{bmatrix} u_t \\ 1 \end{bmatrix} \text{diag}(d_t) \\ 1 \end{bmatrix}, \tag{17}$$

$$\begin{bmatrix} u_{T+F} \\ 1 \end{bmatrix} = K \begin{bmatrix} x_t/z_t \\ y_t/z_t \\ 1 \end{bmatrix}, \tag{18}$$

$$\tilde{m}^B_t(u_{T+F}) = m^B_t(u_t), \tag{19}$$

$$\tilde{d}^B_t(u_{T+F}) = z_t, \tag{20}$$

where u_t is a vector whose entries dictate the pixel locations in m_t which correspond to background object classes and u_{T+F} is the vector which contains the location of these pixels in the target frame at time $T+F$. During this, we maintain the semantic class obtained from m_t and the projected depth of each pixel location. Whenever multiple pixels u_t from an input frame are projected to the same pixel u_{T+F} in the target frame, the depth and the semantic label of the pixel with the smallest depth is kept, as it is closest to the camera.

Given reprojected semantics $\tilde{m}^B_t$ and depths $\tilde{d}^B_t$ from the previous step, the background refinement model is tasked with predicting a final semantic output. This is done by concatenating the input from all frames and feeding them into a semantic

segmentation model, which can be described as

$$\widehat{m}^{\text{Prob}}_{T+F} = \text{BGRef}([\{\tilde{m}^B_t(u_{T+F})\tilde{d}^B_t(u_{T+F})\}_{1:T}])$$
$$\widehat{m}^B_{T+F} = \arg\max_c(\widehat{m}^{\text{Prob}}_{T+F}),$$

where $\widehat{m}^{\text{Prob}}_{T+F} \in \Delta^{H\times W}_{C_{\text{BG}}}$ represents the C_{BG}-dimensional output probability map per pixel, one for each background class, and the final output $\widehat{m}^B_{T+F}$ is obtained per-pixel by choosing the class with the largest probability.

The refinement network is trained using the cross-entropy loss

$$\mathcal{L}_{\text{bf}} \coloneqq \frac{1}{\sum_{x,y}\mathbf{1}^{\text{bg}}_{T+F}[x,y]}\sum_{x,y}\mathbf{1}^{\text{bg}}_{T+F}[x,y]\sum_c m^{B*}_{T+F}(x,y,c)\log\left(\widehat{m}^{\text{Prob}}_{T+F}(x,y)\right). \tag{21}$$

Here, $\mathbf{1}^{\text{bg}}_{T+F}[x,y]$ is an indicator function specifying whether pixel coordinates (x,y) correspond to background semantic classes for frame $T+F$, and $m^{i*}_{T+F}(x,y,c)=1$ if the correct class for pixel (x,y) is c and 0 otherwise. For all experiments presented in this work, we use the specific background prediction model trained by Graber *et al*. [14]. Further implementation details related to model architecture and training can be found in the Appendix of [14].

B. Architecture details for Forecasting Transformer Encoder and Decoder

The feature model f_{Loc} processes input locations $\mathbf{x}^i_t$, appearances $\mathbf{r}^i_t$, instance classes c^i, odometry o_t, and time t to produce an embedding $\bar{\mathbf{x}}^i_{\text{Loc},t}$ which is processed by the transformer FTE. f_{Loc} can be fully specified by the following model components:

$$\mathbf{x}'^i_t = f_b([\mathbf{x}^i_t, \text{onehot}(c^i)]), \tag{22}$$
$$\mathbf{r}'^i_t = \text{AvgPool}(f_f(\mathbf{r}^i_t)), \tag{23}$$
$$\bar{\mathbf{x}}^i_{\text{Loc},t} = f_{\text{e2}}([f_{\text{e1}}([\mathbf{x}'^i_t, \mathbf{r}'^i_t, o_t]), \tau_t]). \tag{24}$$

First, an initial location embedding $\mathbf{x}'^i_t$ is produced, where f_b is a linear layer and *onehot* represents a vector whose c^i-th element is set to one and whose other entries are set to zero. Similarly, initial appearance embedding $\mathbf{r}'^i_t$ is produced, where f_f is a small convolutional network and *AvgPool* averages the result over the spatial dimensions. These two embeddings are concatenated with odometry o_t, passed through linear layer f_{e1}, concatenated with temporal encoding τ_t, and passed through the final linear layer f_{e2}. Specifically, the temporal encoding $\tau^t \in \mathbb{R}^{d_\tau}$ provides information to the model about the temporal location of the given instance in the sequence and whose k-th element is defined as

$$\tau^t(k) = \begin{cases}\sin(t/1000^{k/d_\tau}), & k \text{ is even}\\ \cos(t/1000^{(k-1)/d_\tau}), & k \text{ is odd}\end{cases}. \tag{25}$$

Here d_τ is the size of the temporal encoding and is set to 256 everywhere in this work. All linear layers in f_{Loc} have an output embedding size of 256, and f_f contains two 2D convolutional layers with a kernel size of 3, output channel size of 256, and ReLU activations after each.

The feature model f_{App} produces appearance embedding $\bar{\mathbf{x}}^i_{\text{App},t}$ as a function of the input appearances $\mathbf{r}^i_t$ as well as input time t, and can be fully specified by the following model components:

$$\bar{\mathbf{x}}^i_{\text{App},t} = f_{\text{ae2}}([f_{\text{ae1}}(\mathbf{r}^i_t), \tilde{\tau}_t]), \tag{26}$$

where f_{ae1} is a 3×3 convolutional layer with output dimension 256, f_{ae2} is a 1×1 convolutional layer with output dimension 256, and $\tilde{\tau}_t \in \mathbb{R}^{d_\tau\times 14\times 14}$ is equivalent to τ_t copied across spatial dimensions to match the size of $\mathbf{r}^i_t$.

The location transformer encoder FTE_{Loc} consists of two stacks of transformer encoder modules as originally defined in [42] consisting of layer norm, multi-head self-attention, feed-forward networks, and residual connections. Specifically, all transformers in this work use the Pre-LN construction [48], where the Layer Norm module is placed before the multi-head attention and feed-forward network, as we observed improved convergence. As specified in Sec. 3.2, the multi-head attention modules use both difference attention (Sec. 3.1) and agent-aware attention (Appendix C). The embedding dimension of all

keys, queries, and values as well as the output $\mathbf{h}^i_{\text{Loc},t}$ is 256, the hidden dimension of feedforward modules is 512, the dropout rate used is 0.1, and the number of heads used for multi-head attention is 8.

The appearance transformer encoder FTE_{App} additionally consists of two stacks of transformer encoder modules. However, unlike FTE_{Loc}, the standard dot-product attention formulation is used, and all linear projections in both the multihead attention modules as well as the feedforward network are replaced with 2D convolutional layers with a filter size of 3×3. All embeddings maintain the same spatial dimensions of 14×14 during computation, the channel dimension used is 256, the hidden channel dimension of the feedforward modules is 512, the dropout rate used is 0.1, and the number of heads used for multi-headed attention is 8.

Note, for readability we formulate all models assuming every instance i is present at every input time step $t \in \{1, \ldots, T\}$. However, in practice, some instances will not be present in some input time steps due to occlusions or instances entering/leaving the frame, *i.e.*, there are instances i and input frames t for which $p^i_t = 0$. For all i, t such that $p^i_t = 0$, we do not compute $\bar{\mathbf{x}}^i_{\text{Loc},t}$ or $\bar{\mathbf{x}}^i_{\text{App},t}$ since there are no inputs from which we can compute these. Consequentially, neither FTE_{Loc} nor FTE_{App} receive input representing instance i for time t and thus do not produce encoder representations $\mathbf{h}^i_{\text{Loc},t}$ and $\mathbf{h}^i_{\text{App},t}$ for them.

The decoder location feature model $\tilde{f}_{\text{Loc}}$ produces the feature representation $\tilde{\mathbf{x}}^i_{\text{Loc},t}$ containing information about the most recently predicted location, odometry, and the corresponding instance class. $\tilde{f}_{\text{Loc}}$ can be fully specified by the following model components:

$$\mathbf{x}''^i_t = f_{\text{d1}}([\widehat{\mathbf{x}}^i_{t-1}, \text{onehot}(c^i), o^t), \tag{27}$$

$$\tilde{\mathbf{x}}^i_{\text{Loc},t} = f_{\text{d2}}([\mathbf{x}''^i_t, \tau_t]). \tag{28}$$

First, an initial representation $\mathbf{x}''^i_t$ is computed from the previous location prediction $\widehat{\mathbf{x}}^i_{t-1}$ using linear layer f_{d1}, corresponding instance class c^i, and odometry o_t. This is concatenated with temporal encoding τ_t and passed through a second linear layer f_{d2}. Both f_{d1} and f_{d2} use output dimension equal to 256.

The decoder appearance feature model $\tilde{f}_{\text{App}}$ produces the feature representation $\tilde{\mathbf{x}}^i_{\text{App},t}$ containing information about the most recently predicted appearance. $\tilde{f}_{\text{App}}$ can be fully specified by the following model components:

$$\tilde{\mathbf{x}}^t_{\text{App},t} = f_{\text{ad2}}([f_{\text{ad1}}(\widehat{\mathbf{r}}^i_{t-1}), \tilde{\tau}_t]), \tag{29}$$

where f_{ad1} and f_{ad2} are convolutional layers with the same structure as f_{ae1} and f_{ae2}, respectively.

Both the location and appearance transformer decoders FDE_{Loc} and FDE_{App} use the same construction and hyperparameters as their encoder counterparts. The primary difference is that they are transformer decoders as defined in [42] and hence additionally introduce cross attention layers which operate on the encoder representations $\{\mathbf{h}^i_{\text{Loc},t}\}^{1:N}_{1:T}$ and $\{\mathbf{h}^i_{\text{App},t}\}^{1:N}_{1:T}$, respectively. Output decoder representations $\tilde{\mathbf{h}}^i_{\text{Loc},t}$ and $\tilde{\mathbf{h}}^i_{\text{Loc},t}$ are computed autoregressively; *e.g.*, previous predictions $\{\widehat{\mathbf{x}}^i_t\}^{1:N}_{T:t'-1}$ for times T through $t'-1$ are used to compute the outputs $\{\tilde{\mathbf{h}}^i_{\text{Loc},t'}\}$. These embeddings are then used to produce $\{\widehat{\mathbf{x}}^i_{t'}\}^{1:N}$ for time t', and these new predictions are fed back into the model to produce output for the next time step $t'+1$, and so on. Decoder attention is masked to maintain causality, *i.e.*, embeddings representing a given time t are prevented from attending to representations for future time steps $t' > t$.

f_{LocOut}, f_{POut}, and f_{vel} are all 3-layer multilayer perceptrons with hidden sizes $[512, 256]$ and ReLU activations. f_{AppOut} is a 3×3 convolutional layer.

C. Agent-aware Attention

Due to their permutation-invarance with respect to their inputs, transformers do not have the inherent capacity to reason about the identity of the entities corresponding to input trajectories. To address this problem, Yuan *et al*. [54] introduced agent-aware attention. This approach allows transformers to encode the identity of its inputs within the model, which makes it easier for these models to reason about the trajectories of individual entities and leads to better forecasting performance.

Let $\mathbf{X}_{\text{self}} \in \mathbb{R}^{M_1 \times d}$ and $\mathbf{X}_{\text{other}} \in \mathbb{R}^{M_2 \times d}$ of lengths M_1 and M_2, respectively, be the input sequences with embedding dimension d. For self-attention, both input sequences are the same and represent the input trajectories of a number of agents, while for cross-attention, the first input sequence corresponds to a future trajectory forecast and the second corresponds to

input trajectories. The agent-aware attention output $\mathbf{Y} \in \mathbb{R}^{M_1 \times d}$ is then computed as

$$\mathbf{Z} = \mathbf{M} \odot (\mathbf{Q}_{\text{agent}} \mathbf{K}_{\text{agent}}^T) + (1 - \mathbf{M}) \odot (\mathbf{Q}_{\text{context}} \mathbf{K}_{\text{context}}^T), \tag{30}$$

$$\mathbf{Y} = \text{softmax}\left(\mathbf{Z}/\sqrt{d}\right)\mathbf{V}, \tag{31}$$

where $\odot$ represents element-wise multiplication. Specifically, agent-aware attention first computes two sets of keys $\mathbf{K}_{\text{agent}} = f_{K,\text{agent}}(\mathbf{X}_{\text{other}})$, $\mathbf{K}_{\text{context}} = f_{K,\text{context}}(\mathbf{X}_{\text{other}})$ and queries $\mathbf{Q}_{\text{agent}} = f_{Q,\text{agent}}(\mathbf{X}_{\text{self}})$, $\mathbf{Q}_{\text{context}} = f_{Q,\text{context}}(\mathbf{X}_{\text{self}})$ from the original inputs. It then computes two sets of attention scores from the *agent* keys/queries and from the *context* keys/queries and selects between them using mask $\mathbf{M} \in \{0,1\}^{M_1 \times M_2}$. This mask encodes identity information: $\mathbf{M}_{ij} = 1$ if entity i in the first input sequence and entity j in the second input sequence correspond to the same agent, and $\mathbf{M}_{ij} = 0$ otherwise. In other words, two sets of attention parameters are computed, and one set is used for input pairs corresponding to the same agent while the other is used for all pairs corresponding to different agents, *i.e.*, the context for this agent. Value aggregation proceeds as in standard attention from this step.

We additionally use agent-aware attention within the difference attention module defined in Sec. 3.1. This is implemented in a similar fashion, where separate attention parameters are computed for input pairs corresponding to the same agent and for input pairs corresponding to different agents. We formally specify this as

$$\mathbf{Z} = \mathbf{M} \odot \left(\mathbf{Q}_{\text{agent}} \mathbf{K}_{R,\text{agent}}^T - \mathbf{1}_{M_1 \times 1} \text{diag}\left(\mathbf{K}_{B,\text{agent}} \mathbf{K}_{R,\text{agent}}^T\right)^T\right) + \tag{32}$$

$$(1 - \mathbf{M}) \odot \left(\mathbf{Q}_{\text{context}} \mathbf{K}_{R,\text{context}}^T - \mathbf{1}_{M \times 1} \text{diag}\left(\mathbf{K}_{B,\text{context}} \mathbf{K}_{R,\text{context}}^T\right)^T\right), \tag{33}$$

$$\mathbf{Y} = \text{softmax}\left(\mathbf{Z}/\sqrt{d}\right)\mathbf{V}_O - \mathbf{V}_S, \tag{34}$$

with $\mathbf{K}_{R,\text{agent}} = f_{K,R,\text{agent}}(\mathbf{X}_{\text{other}})$, $\mathbf{K}_{R,\text{context}} = f_{K,R,\text{context}}(\mathbf{X}_{\text{other}})$, $\mathbf{K}_{B,\text{agent}} = f_{K,B,\text{agent}}(\mathbf{X}_{\text{other}})$, $\mathbf{K}_{B,\text{context}} = f_{K,B,\text{context}}(\mathbf{X}_{\text{other}})$, $\mathbf{V}_O = f_{V_O}(\mathbf{X}_{\text{other}})$, and $\mathbf{V}_S = f_{V_S}(\mathbf{X}_{\text{self}})$.

D. Losses for Foreground Forecasting

The loss used by the foreground forecasting model are

$$\mathcal{L}_{\text{FG}} = \mathcal{L}_{\text{Loc}} + \mathcal{L}_{\text{P}} + \mathcal{L}_{\text{App}} + \mathcal{L}_{\text{Vel}}. \tag{35}$$

The location loss $\mathcal{L}_{\text{Loc}}$ trains the bounding box predictions $\widehat{\mathbf{x}}_{\text{Box,t}}^i \coloneqq [\widehat{x}_{0,t}, \widehat{y}_{0,t}, \widehat{x}_{1,t}, \widehat{y}_{1,t}]$ and depth predictions $\widehat{d}_t^i$ to match the target boxes $\mathbf{x}_{\text{Box,t}}^{*i}$ and depths d_t^*. This is specified as

$$\mathcal{L}_{\text{Loc}} \coloneqq \frac{1}{\sum_{i=1}^{N}\sum_{t=T+1}^{T+F} p_t^{*i}} \sum_{i=1}^{N}\sum_{t=T+1}^{T+F} p_t^{*i}\left(\lambda_1 \text{SmoothL1}(\widehat{\mathbf{x}}_{\text{Box},t}^i, \mathbf{x}_{\text{Box},t}^{*i}) + \lambda_2 \text{SmoothL1}(\widehat{d}_t^i, d_t^{i*}) + \lambda_3 \text{IoU}(\widehat{\mathbf{x}}_{\text{Box},t}^i, \mathbf{x}_{\text{Box},t}^{*i})\right), \tag{36}$$

where p_t^{*i} is ground-truth presence, *i.e.*, equals 1 if instance i is present in frame t and 0 otherwise, IoU is bounding box intersection-over-union, SmoothL1 is the function

$$\text{SmoothL1}(\mathbf{a}, \mathbf{b}) \coloneqq \sum_j \text{SmoothL1Fn}(\mathbf{a}_j, \mathbf{b}_j), \tag{37}$$

$$\text{SmoothL1Fn}(a, b) \coloneqq \begin{cases} \frac{1}{2}(a-b)^2, & \text{if } |a-b| < 1, \\ |a-b| - \frac{1}{2} & \text{otherwise} \end{cases}, \tag{38}$$

and coefficients $\lambda_1 = 1$, $\lambda_2 = 10$, $\lambda_3 = 100$ are used to balance the magnitudes of the losses.

The presence loss $\mathcal{L}_{\text{P}}$ trains the presence predictions $\widehat{p}_t^i \in \mathbb{R}$ to correctly indicate whether a given instance i is present in frame t, and is computed as

$$\mathcal{L}_{\text{P}} \coloneqq \frac{\lambda_4}{NF} \sum_{i=1}^{N}\sum_{t=T+1}^{T+F} p_t^{*i} \log \sigma(\widehat{p}_t^i) + (1 - p_t^{*i}) \log(1 - \sigma(\widehat{p}_t^i)), \tag{39}$$

where σ is the sigmoid function and $\lambda_4 = 10$.

The appearance loss $\mathcal{L}_{\text{App}}$ trains the appearance predictions $\widehat{\mathbf{r}}_t^i$ for instance i at frame t to match the target features $\mathbf{r}_t^{*i}$ extracted for this instance at frame t, and consists of the mean-squared error of the features for all valid instance/time pairs, *i.e.*,

$$\mathcal{L}_{\text{App}} \coloneqq \frac{\lambda_5}{\sum_{i=1}^{N}\sum_{t=T+1}^{T+F} J p_t^{*i}} \sum_{i=1}^{N}\sum_{t=T+1}^{T+F}\sum_{j=1}^{J} p_t^{*i}(\widehat{\mathbf{r}}_{j,t}^i - \mathbf{r}_{j,t}^{*i})^2, \tag{40}$$

where j indexes over all spatial dimensions of the feature tensors, $J = 256 \times 14 \times 14$ is the total number of elements of the feature tensors, and $\lambda_5 = 10$.

The encoder velocity loss $\mathcal{L}_{\text{Vel}}$ trains the velocity predictions $\widehat{\mathbf{v}}_{\text{E},t}^i \in \mathbb{R}^4$ to match the ground-truth velocities $\mathbf{v}_t^{*i} \coloneqq \mathbf{x}_{t+1}^{*i} - \mathbf{x}_t^{*i}$, and is computed as

$$\mathcal{L}_{\text{Vel}} \coloneqq \frac{\lambda_6}{\sum_{i=1}^{N}\sum_{t=1}^{T} p_t^{*i} p_{t+1}^{*i}} \sum_{i=1}^{N}\sum_{t=1}^{T} p_t^{*i} p_{t+1}^{*i} \text{SmoothL1}(\widehat{\mathbf{v}}_{\text{E},t}^i, \mathbf{v}_t^{*i}), \tag{41}$$

where $\lambda_6 = 1$.

E. Depth Completion Model

The depth completion model operates on noisy and incomplete reprojected background depth $\tilde{d}^B$ along with depth mask Q and background class probabilities $\widehat{m}^{\text{Prob}}$ and produces depth maps $\widehat{d}^{\text{B}}_{\text{Fill}}$ and $\widehat{d}^{B}_{\text{Bias}}$. This model can be formally represented using the following components:

$$d_1 = f_{\text{dc1}}([\tilde{d}^B, Q, \widehat{m}^{\text{Prob}}]), \tag{42}$$

$$d_2 = d_1 + \text{Upsample}(f_{dc2}(\text{Downsample}(d_1)), \tag{43}$$

$$\widehat{d}^{B}_{\text{Fill}} = f_{\text{fill}}(d_2), \tag{44}$$

$$\widehat{d}^{B}_{\text{Bias}} = f_{\text{Bias}}(d_2). \tag{45}$$

First, reprojected background depth $\tilde{d}^B$, depth mask Q, and predicted background class probabilities $\widehat{m}^{\text{Prob}}$ are concatenated together and processed with convolutional layer f_{dc1} which uses a kernel size of 3 and has output channel dimension 32. The output of this, d_1, is downsampled by a factor of 2 using bilinear interpolation, fed into convolutional network f_{dc2}, upsampled to the original resolution using bilinear interpolation, and added with d_1 to produce the second intermediate output d_2. f_{dc2} contains 2 convolutional layers with a kernel size of 3, output channel dimension of 32, and a ReLU activation between them. The outputs $\widehat{d}^{\text{B}}_{\text{Fill}}$ and $\widehat{d}^{B}_{\text{Bias}}$ are then obtained from d_2 using convolutional networks f_{fill} and f_{bias}, respectively. Both of these networks contain two convolutional layers with a ReLU activation between them, where the first layer uses a kernel size of 3 and an output channel size of 32 and the second layer uses a kernel size of 1. The final background depth estimate $\widehat{d}^B$ is obtained from outputs $\widehat{d}^{\text{B}}_{\text{Fill}}$ and $\widehat{d}^{B}_{\text{Bias}}$ as specified in Eq. (13).

F. Additional Implementation Details

The overall approach is trained in two stages: first, the foreground prediction model is trained; afterwards, the corresponding parameters are frozen, and the refinement model is trained.

The foreground model is trained for 48000 steps using the ADAM optimizer; the initial learning rate is set to 10^{-4}, and it is lowered to 10^{-5} after 36000 optimization steps. All odometries o_t are normalized by subtracting the training data set mean and then dividing by training data set standard deviation before being used as input. All location inputs $\mathbf{x}_t^i$ are normalized to lie within $[-1, 1]$; furthermore, location outputs $\widehat{\mathbf{x}}_t^i$ are made at this normalized scale and unnormalized before being used at later stages. During training and inference, forecasts are only predicted for instances present in the most recent input frame, *i.e.*, for instances i such that $p_T^i = 1$. During training of the foreground model, ground-truth future odometry is used. During evaluation, unless otherwise noted, the egomotion estimation module described by Graber *et al*. [14] was used to obtain future odometry which was used as input. We use the same odometry representation as Graber *et al*. [14] consisting of a five-dimensional vector containing speed and yaw rate of the ego-vehicle at time t as well its top-down displacement and angular displacement between steps t and $t-1$.

Ablation 4 in this work uses odometry during inference that was obtained using ORB-SLAM3 [2]. This was run using stereo images, where each sequence of 30 frames was treated as its own SLAM session providing 6-dof poses for all frames in the sequence.

The refinement model is trained for 24000 steps using the ADAM optimizer; the initial learning rate is set to 10^{-4}, and it is lowered to 10^{-5} after 18000 optimization steps. During training, the inputs are scaled to a spatial resolution of $\frac{H}{4} \times \frac{W}{4}$, and the loss is additionally computed at this scale. During inference, inputs are scaled to the final spatial resolution, *i.e*., $H \times W$.

To process a Cityscapes sequence, the model needs 560 ms on average using an NVIDIA A6000, which is on par with the 700 ms required by Graber *et al*. [14]. This can be significantly reduced by further engineering effort.

G. Additional AIODrive Details

The AIODrive sequences are annotated using 23 object classes. To facilitate comparison against results on the Cityscapes dataset, we only train and evaluate using background classes which are also present in Cityscapes. This leaves 11 background "stuff" classes and 2 foreground "things" classes (the only annotated "things" instances in AIODrive are "vehicles" and "pedestrians"). As annotations are only provided for the trainval dataset, we split this into a training dataset containing all annotated sequences for towns 1 through 5 and a validation dataset containing all annotated sequences for town 6. We use 5 frames as input and forecast the 5th frame into the future, corresponding to 0.5 seconds of input and a 0.5 second forecast (which is comparable to the Cityscapes mid-term setting).

During both training and evaluation, we only consider instances whose masks have an area of at least 400 pixels in an attempt to filter out distant, imperceptible instances. For evaluation, we use non-overlapping sequences of 10 frames from each validation sequence. Additionally, the data contains some periods of time with little to no motion, which skews the evaluation metrics artificially high. To ensure that the metrics properly capture the ability of the models to anticipate motion, we filter out validation sequences where the recording vehicle is moving less than 1 m/s at all points in the input sequence and where at least half of the instance mask centers move less than 10 pixels. This leaves 814 sequences with motion for evaluation purposes. To ensure that the tracking-based metrics can be computed, we use ground-truth instance bounding boxes and ids as input to the forecasting models.

The base semantic and instance segmentation models are the same as that used for Cityscapes, *i.e*., MaskRCNN [16] for instance segmentation and Panoptic Deeplab [3] for semantic segmentation. For both, we initialize from the Cityscapes pre-trained model and finetune on AIODrive. For the models that use predicted depth, we use Cascade-Stereo [15] on the stereo input images. We do not finetune the depth model on this dataset.

H. Cityscapes Instance Segmentation

We also evaluate our Cityscapes-trained model on instance segmentation. Here, we consider only 'things' instances during evaluation, and hence we disregard the pixels corresponding to the 'stuff' classes.

Metrics. We evaluate instance segmentation using the standard metrics [7]: 1) *Average Precision* (AP) computes true positives using a number of overlap thresholds, averages over these thresholds, and then averages over classes; 2) AP50 computes average precision with an overlap threshold of 0.5 and then averages across classes.

Baselines. We compare against the baselines presented by Graber *et al*. [14]. F2F is introduced by Luc *et al*. [27] and predicts the features of a future scene using a convolutional model. It then obtains instances by passing these features through MaskRCNN heads. IndRNN-Stack is the independent RNN and stacking model by Graber *et al*. [14]. PFA, introduced by Lin *et al*. [25], compresses input feature pyramids into a low-resolution feature map for forecasting.

Results. The results for this task are presented in Tab. 4. We outperform F2F and IndRNN-Stack in the mid-term setting but PFA performs better. This is to be expected because PFA was directly trained on instance segmentation while we directly apply the model trained on panoptic segmentation, *i.e*., we don't retrain our model specifically for instance segmentation.

I. Cityscapes Semantic Segmentation

Following prior work [14], we also evaluate our model on semantic segmentation forecasting. In this context, we do not care about specific instances. Hence, for each pixel, we discard all predicted identity information.

Metrics. Semantic segmentation forecasting is evaluated using the standard *intersection over union* (IoU) metric computed between predictions and ground truth per class and averaged over classes. IoU (MO), meanwhile, computes an average IoU over 'things' classes only.

	Short term: $\Delta t = 3$		Mid term: $\Delta t = 9$	
	AP	AP50	AP	AP50
Oracle	34.6	57.4	34.6	57.4
Last seen frame	8.9	21.3	1.7	6.6
F2F [27]	19.4	**39.9**	7.7	19.4
IndRNN-Stack [14]	17.8	38.4	10.0	22.3
PFA [25]	**24.9**	**48.7**	**14.8**	**30.5**
Ours	19.9	39.9	11.2	25.2

Table 4. **Instance segmentation forecasting on the Cityscapes Validation dataset.** Higher is better for all metrics.

	Short term: $\Delta t = 3$		Mid term: $\Delta t = 9$	
Accuracy (mIoU)	All	MO	All	MO
Oracle	80.6	81.7	80.6	81.7
Copy last	59.1	55.0	42.4	33.4
Bayesian S2S [1]	65.1	/	51.2	/
DeformF2F [36]	65.5	63.8	53.6	49.9
LSTM M2M [40]	67.1	65.1	51.5	46.3
F2MF [37]	69.6	67.7	57.9	54.6
IndRNN-Stack [14]	67.6	60.8	58.1	52.1
PFA [25]	**71.1**	**69.2**	**60.3**	**56.7**
Ours	67.9	61.2	58.1	51.7

Table 5. **Semantic forecasting results on the Cityscapes validation dataset**. Baseline numbers, besides oracle and copy last, are from [37]. Higher is better for all metrics. Our model exploits stereo and odometry, which are provided by typical autonomous vehicle setups and are included in Cityscapes.

Baselines. Many of the baselines operate by predicting the features of a future scene [1, 25, 36, 37]. LSTM M2M [40] warps input semantics using a predicted optical flow between the most recent frame and the target frame. Note that these approaches do not use depth inputs, and all except Bayesian S2S [1] do not use egomotion as input.

Results. The results for this task are given in Tab. 5. We outperform IndRNN-Stack by a small margin in the short-term setting, and have comparable results in the mid-term setting. We additionally outperform most other baselines. Note that this metric does not care about the boundaries between individual instances and hence weights some types of errors differently than the other metrics we use. These other metrics more properly evaluate whether specific instances are localized in the correct places, which we argue better captures the goals of forecasting. Note that PFA is directly trained on semantic segmentation forecasting while our approach was trained on forecasting of panoptic segmentations.

J. Additional Cityscapes Visualizations

Fig. 6 presents a visual comparison between our approach and IndRNN-Stack for the short-term setting for the sequences which were shown for the mid-term setting in Fig. 3. We present additional visualizations for the mid-term setting in Fig. 7 and for the short term setting in Fig. 8.

K. Additional Cityscapes Metrics

Tabs. 6 to 8 present metrics computed for the Cityscapes test dataset using the mid-term setting for panoptic, instance, and semantic segmentation forecasting, respectively. We outperform all other approaches for panoptic and instance segmentation forecasting on the test data. On semantic segmentation, we outperform IndRNN-Stack on the test data, whereas F2MF [37] outperforms our approach. However, note that the F2MF model used for test evaluation was trained on both the training and validation datasets, while the other models were trained only on the training data.

Tabs. 9 to 14 contain the per-class breakdown of all panoptic segmentation metrics shown in Tab. 1. The results shown in Tab. 1 consist of the average of these metrics taken over the values obtained for each individual class. Our model is better

	All			Things			Stuff		
	PQ	SQ	RQ	PQ	SQ	RQ	PQ	SQ	RQ
Flow	25.6	70.1	34.0	12.4	66.3	18.1	35.3	72.9	45.5
Hybrid [40] (bg) and [27] (fg)	29.4	69.8	38.5	18.0	67.2	25.7	37.6	71.6	47.8
IndRNN-Stack	35.7	72.0	46.5	24.0	69.0	33.7	44.2	74.2	55.8
Ours	**36.9**	**72.7**	**48.0**	**26.7**	**70.3**	**37.0**	**44.4**	**74.4**	**55.9**

Table 6. **Panoptic segmentation forecasting evaluated on the Cityscapes test set, mid-term.** Higher is better for all metrics.

	AP	AP50
F2F [27]	6.7	17.5
IndRNN-Stack	8.4	19.8
Ours	**9.9**	**20.7**

Table 7. **Instance segmentation forecasting on the Cityscapes Test dataset, mid-term.** Higher is better for all metrics.

Accuracy (mIoU)	All	MO
F2MF [37]*	59.1	56.3
IndRNN-Stack	57.7	48.8
Ours	58.3	50.0

Table 8. **Semantic segmentation forecasting results on the Cityscapes test dataset.** Baseline numbers, are from [37]; the * indicates training on both train and validation data. Higher is better for all metrics.

	road	sidewalk	building	wall	fence	pole	traffic light	traffic sign	vegetation	terrain	sky	person	rider	car	truck	bus	train	motorcycle	bicycle	mean
Deeplab (Oracle)†	97.9	78.2	88.5	29.4	38.9	60.0	55.6	74.5	89.5	36.1	87.9	50.8	46.4	67.3	51.5	66.6	37.8	44.2	44.1	60.3
Deeplab (Last seen frame)	94.3	52.4	71.1	11.3	19.4	6.1	12.9	15.0	72.1	16.9	72.7	10.3	8.0	29.6	35.1	51.7	24.2	9.8	7.9	32.7
Flow	95.6	61.5	79.8	17.3	**28.6**	8.7	26.2	36.8	80.7	**26.9**	79.7	21.0	14.0	43.4	40.6	56.8	26.7	23.2	18.7	41.4
Hybrid [40] (bg) and [27] (fg)	**96.2**	63.4	81.4	23.1	23.7	7.1	19.1	36.9	82.3	20.3	79.8	26.8	21.8	46.4	**42.2**	60.0	41.4	25.6	22.5	43.2
IndRNN-Stack	**96.2**	66.1	83.5	**26.1**	27.4	31.7	**37.0**	49.9	**84.8**	26.1	82.0	31.8	31.5	48.8	**42.2**	61.2	47.0	31.4	27.3	49.0
Ours	**96.2**	**66.3**	**83.8**	25.9	27.4	**34.4**	**37.0**	**50.3**	**84.8**	26.5	**82.1**	**34.9**	**36.7**	**51.2**	41.2	**63.1**	**47.6**	**32.4**	**32.0**	**50.2**

Table 9. **Per-class results for Panoptic Quality on Cityscapes validation dataset (short-term).**

on average for every metric than all other approaches, and it is additionally better than prior approaches for every metric for most classes.

L. Code Details

All models are implemented using PyTorch v. 1.10.0 [2], which is made available for use with a custom BSD-style license.[3] We additionally use the Detectron2 framework (version 0.4.1)[4], which is released under the Apache 2.0 license.[5] Code implementing our models and experiments can be found at https://github.com/cgraber/psf-diffattn.

M. Potential Negative Societal Impact

One of the primary applications of this work is to better enhance the ability of autonomous agents to anticipate the future and respond appropriately to a dynamic environment. In this context, problems can arise if an agent makes a decision based on a faulty prediction – for example, if a self-driving car does not anticipate a pedestrian stepping into the street, it could unintentionally hurt the pedestrian if they step out in front of the car. For such a system, the consequence of prediction errors can be injury or death. It is thus critical that appropriate care be taken before deployment of such a system to ensure that not only are prediction errors sufficiently low across a variety of environments but also that proper failsafes are put in place to minimize the negative consequences of acting upon a misprediction.

[2] https://pytorch.org/
[3] https://github.com/pytorch/pytorch/blob/v1.10.0/LICENSE
[4] https://github.com/facebookresearch/detectron2
[5] https://github.com/facebookresearch/detectron2/blob/v0.4.1/LICENSE

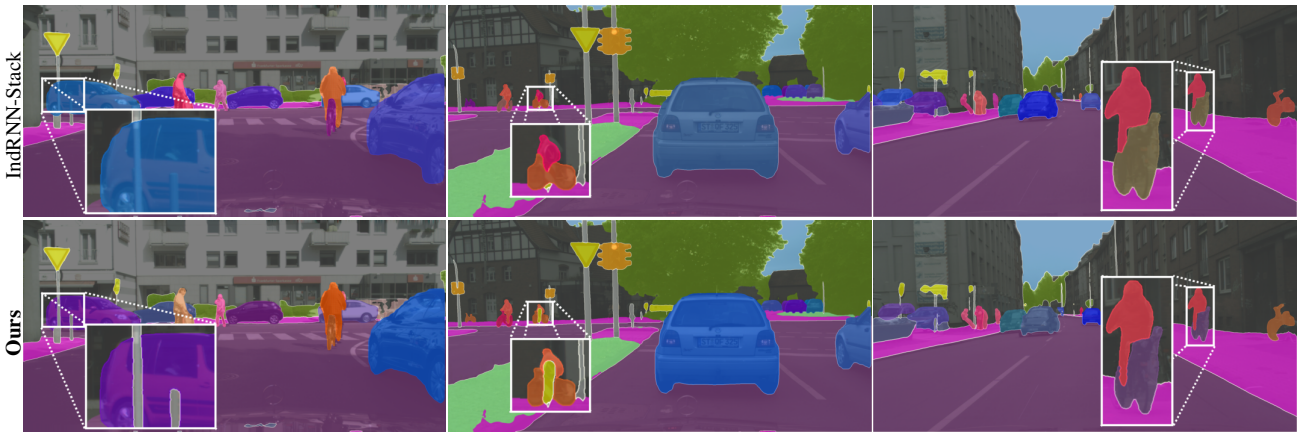

Figure 6. Short-term panoptic segmentation forecasts on Cityscapes.

	road	sidewalk	building	wall	fence	pole	traffic light	traffic sign	vegetation	terrain	sky	person	rider	car	truck	bus	train	motorcycle	bicycle	mean
Deeplab (Oracle)†	97.9	78.2	88.5	29.4	38.9	60.0	55.6	74.5	89.5	36.1	87.9	50.8	46.4	67.3	51.5	66.6	37.8	44.2	44.1	60.3
Deeplab (Last seen frame)	90.4	32.5	57.6	7.6	10.6	4.6	8.9	7.4	55.1	8.8	57.3	5.3	2.5	13.2	19.2	27.3	10.1	4.7	3.0	22.4
Flow	90.5	35.8	66.2	7.7	15.0	4.6	11.9	11.1	65.6	11.6	64.4	5.9	2.5	19.0	21.5	27.7	13.5	11.8	5.3	25.9
Hybrid [40] (bg) and [27] (fg)	93.2	44.9	70.5	12.4	14.8	1.2	8.0	10.8	69.7	13.9	67.2	8.0	4.5	27.3	33.5	41.7	27.9	8.3	6.1	29.7
IndRNN-Stack	**93.9**	50.8	76.4	**18.2**	19.9	8.7	18.7	28.5	**77.0**	**18.6**	**72.7**	16.2	12.0	33.3	36.1	53.0	**29.8**	14.1	12.6	36.3
Ours	**93.9**	**50.9**	**76.5**	18.1	**20.8**	**9.3**	**18.8**	**28.6**	**77.0**	**18.6**	**72.7**	**19.9**	**14.6**	**39.5**	**38.8**	**56.9**	26.2	**18.6**	**14.5**	**37.6**

Table 10. **Per-class results for Panoptic Quality on Cityscapes validation dataset (mid-term).**

	road	sidewalk	building	wall	fence	pole	traffic light	traffic sign	vegetation	terrain	sky	person	rider	car	truck	bus	train	motorcycle	bicycle	mean
Deeplab (Oracle)†	98.0	85.6	90.5	74.3	74.8	69.7	73.5	80.1	90.9	75.7	92.6	76.0	70.8	84.2	88.4	90.8	87.6	73.8	72.1	81.5
Deeplab (Last seen frame)	94.4	71.5	78.8	65.4	65.6	**67.0**	**68.3**	67.4	77.8	67.7	83.0	64.4	60.1	69.2	74.7	76.7	75.7	62.7	63.4	71.3
Flow	95.6	76.0	83.2	68.5	68.3	65.0	65.9	67.3	83.4	69.1	86.6	65.6	61.4	75.8	77.5	80.0	74.1	66.1	64.4	73.4
Hybrid [40] (bg) and [27] (fg)	**96.3**	**77.2**	84.9	70.0	69.0	59.5	63.6	65.9	84.6	70.8	86.5	66.8	61.9	77.2	80.3	83.1	**80.5**	65.6	63.8	74.1
IndRNN-Stack	**96.3**	77.0	86.3	71.1	69.4	61.4	65.4	70.6	**86.6**	**71.3**	**88.3**	67.7	63.8	77.7	81.4	81.4	74.8	67.6	65.8	74.9
Ours	**96.3**	77.1	**86.4**	**71.4**	**69.8**	61.7	65.4	**70.8**	**86.6**	70.9	**88.3**	**69.4**	**66.4**	**78.9**	**81.7**	**84.1**	77.9	**68.1**	**67.2**	**75.7**

Table 11. **Per-class results for Segmentation Quality on Cityscapes validation dataset (short-term).**

	road	sidewalk	building	wall	fence	pole	traffic light	traffic sign	vegetation	terrain	sky	person	rider	car	truck	bus	train	motorcycle	bicycle	mean
Deeplab (Oracle)†	98.0	85.6	90.5	74.3	74.8	69.7	73.5	80.1	90.9	75.7	92.6	76.0	70.8	84.2	88.4	90.8	87.6	73.8	72.1	81.5
Deeplab (Last seen frame)	90.7	68.2	72.6	63.4	62.4	**66.1**	**72.7**	**73.0**	71.2	64.0	77.3	63.7	61.3	66.8	62.9	70.8	74.3	56.4	64.4	68.5
Flow	90.8	68.6	76.0	66.1	64.1	64.1	69.0	67.2	75.0	64.5	78.5	63.5	60.4	69.1	70.2	74.3	**75.8**	60.2	63.0	69.5
Hybrid [40] (bg) and [27] (fg)	93.3	69.7	77.9	66.6	65.3	59.9	62.9	61.9	76.9	65.1	79.6	63.7	58.4	71.5	72.6	72.2	73.7	62.1	60.6	69.1
IndRNN-Stack	94.1	71.3	81.5	68.4	**66.8**	59.0	64.1	65.1	80.9	68.1	**83.0**	64.3	**61.4**	73.4	**76.9**	76.1	74.4	62.5	62.9	71.3
Ours	**94.2**	**71.5**	**81.7**	**69.1**	66.1	60.1	64.2	65.7	**81.0**	**68.5**	**83.0**	**64.7**	61.1	**74.7**	76.5	**79.4**	67.7	**63.6**	**64.7**	**71.4**

Table 12. **Per-class results for Segmentation Quality on Cityscapes validation dataset (mid-term).**

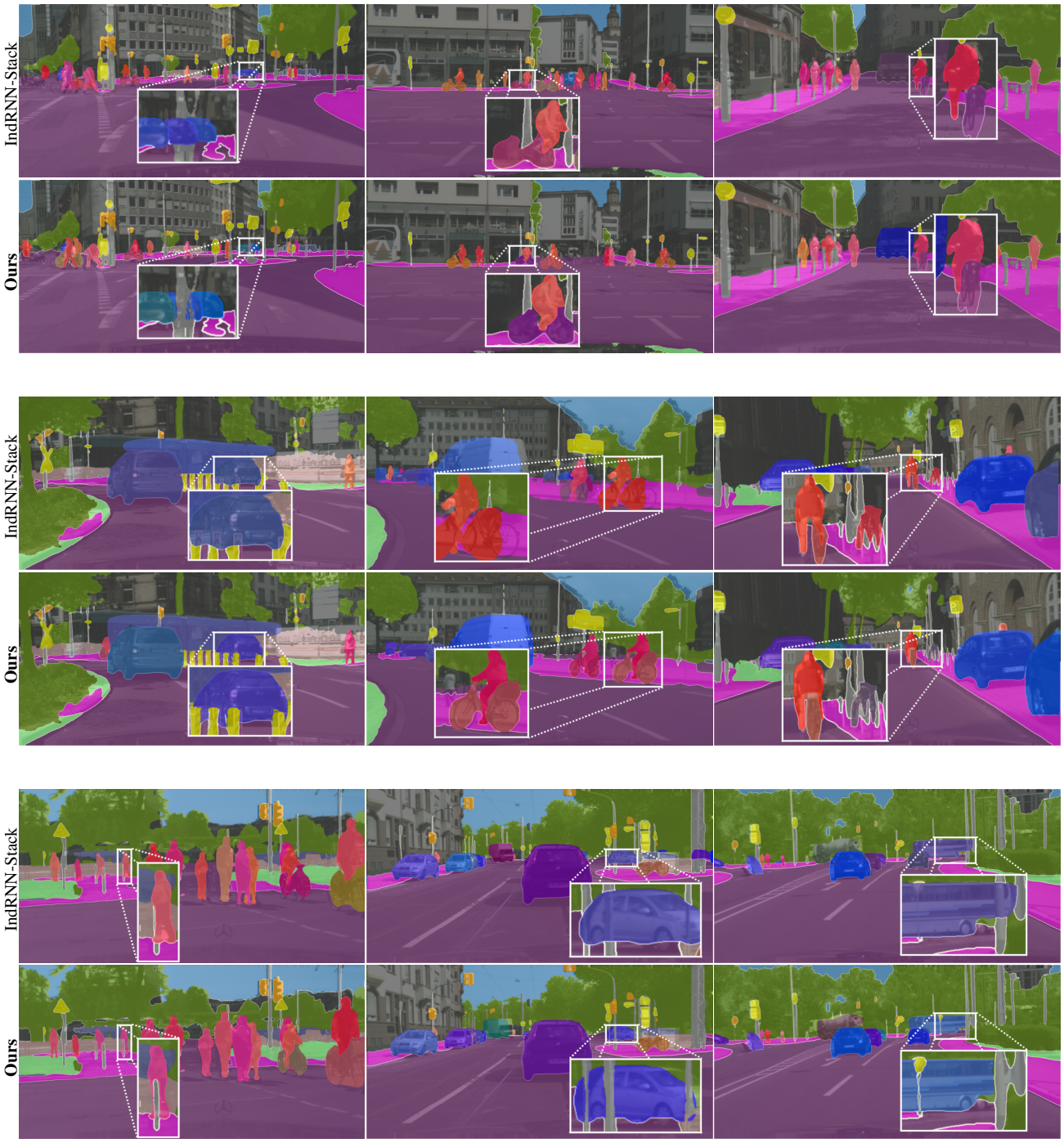

Figure 7. Additional mid-term visualizations.

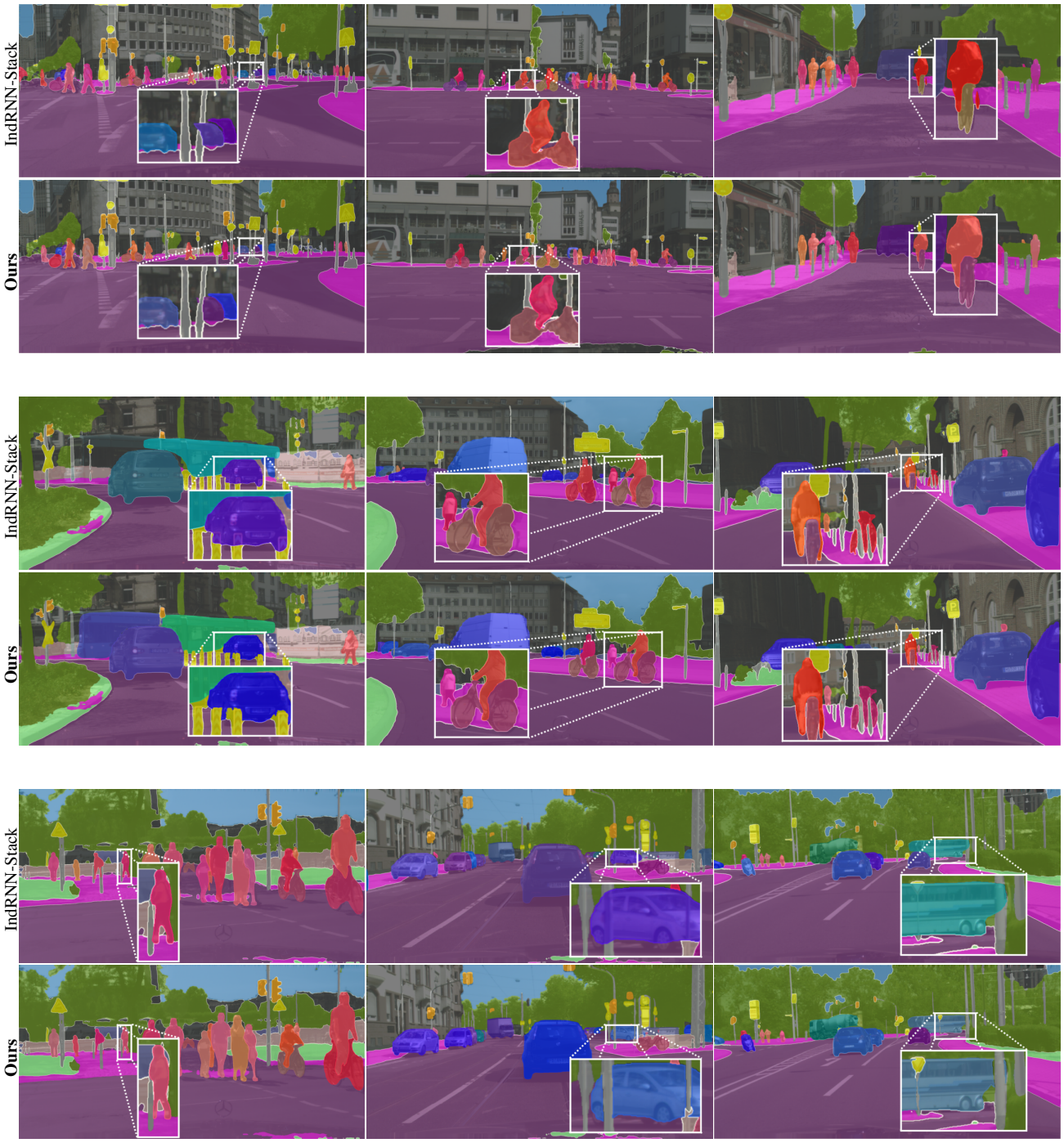

Figure 8. Additional short-term visualizations.

	road	sidewalk	building	wall	fence	pole	traffic light	traffic sign	vegetation	terrain	sky	person	rider	car	truck	bus	train	motorcycle	bicycle	mean
Deeplab (Oracle)†	99.9	91.3	97.8	39.5	52.1	86.1	75.6	93.0	98.5	47.7	94.9	66.9	65.5	80.0	58.2	73.4	43.1	59.9	61.2	72.9
Deeplab (Last seen frame)	**99.9**	73.4	90.2	17.3	29.7	9.1	18.9	22.2	92.7	25.0	87.6	16.0	13.2	42.8	47.0	67.4	32.0	15.6	12.4	42.7
Flow	**99.9**	81.0	95.9	25.2	**41.9**	13.4	39.7	54.7	96.8	**39.0**	92.0	32.1	22.8	57.3	52.4	71.0	36.0	35.1	29.0	53.4
Hybrid [40] (bg) and [27] (fg)	**99.9**	82.1	95.8	33.0	34.4	11.9	30.0	56.0	97.3	28.8	92.2	40.1	35.3	60.2	**52.6**	72.2	51.4	39.1	35.3	55.1
IndRNN-Stack	**99.9**	85.8	96.7	**36.7**	39.4	51.6	**56.6**	70.7	**97.9**	36.6	**92.9**	47.0	49.3	62.9	51.9	**75.2**	**62.9**	46.3	41.5	63.3
Ours	**99.9**	**86.0**	**97.0**	36.2	39.3	**55.7**	**56.6**	**71.1**	**97.9**	37.4	**92.9**	**50.3**	**55.3**	**64.9**	50.4	75.0	61.1	**47.5**	**47.6**	**64.3**

Table 13. **Per-class results for Recognition Quality on Cityscapes validation dataset (short-term).**

	road	sidewalk	building	wall	fence	pole	traffic light	traffic sign	vegetation	terrain	sky	person	rider	car	truck	bus	train	motorcycle	bicycle	mean
Deeplab (Oracle)†	99.9	91.3	97.8	39.5	52.1	86.1	75.6	93.0	98.5	47.7	94.9	66.9	65.5	80.0	58.2	73.4	43.1	59.9	61.2	72.9
Deeplab (Last seen frame)	99.7	47.6	79.3	12.1	17.0	7.0	12.2	10.1	77.4	13.7	74.1	8.3	4.2	19.8	30.6	38.5	13.6	8.3	4.7	30.4
Flow	99.7	52.2	87.1	11.7	23.4	7.2	17.3	16.5	87.4	18.0	82.1	9.2	4.2	27.5	30.6	37.3	17.8	19.6	8.4	34.6
Hybrid [40] (bg) and [27] (fg)	**99.9**	64.5	90.4	18.7	22.7	2.1	12.7	17.4	90.5	21.4	84.4	12.5	7.7	38.2	46.2	57.9	37.8	13.4	10.1	39.4
IndRNN-Stack	99.7	**71.2**	**93.7**	**26.6**	29.8	14.7	29.2	**43.8**	**95.1**	**27.3**	**87.6**	25.2	19.5	45.4	47.0	69.6	**40.0**	22.6	20.0	47.8
Ours	99.7	71.1	**93.7**	26.2	**31.4**	**15.5**	**29.3**	43.5	**95.1**	27.1	**87.6**	**30.8**	**23.9**	**52.9**	**50.7**	**71.7**	38.7	**29.2**	**22.4**	**49.5**

Table 14. **Per-class results for Recognition Quality on Cityscapes validation dataset (mid-term)**.