Supplementary Material for PlanT: Explainable Planning Transformers via Object-Level Representations

Abstract: In this **supplementary document**, we first provide additional methodological details to elaborate on the descriptions from the main paper. Next, we provide information on the implementation of our baselines as well as a detailed description of the model configurations of PlanT. Finally, we provide additional experimental results. The **supplementary video** contains qualitative visualizations of the object relevance for both PlanCNN and PlanT. While PlanCNN does not give temporally consistent visualizations of the object relevance, PlanT reliably identifies relevant objects in dense traffic via its learned attention weights.

1 Methodological Details

In this section, we provide additional details regarding our task formulation, architecture, representation, and metrics.

1.1 Task Formulation

We train all models via supervised learning. An expert agent (taken from Chitta et al. [1]) is first executed to collect a dataset, $\mathcal{D} = \{(\mathcal{X}^i, \mathcal{W}^i)\}_{i=1}^{Z}$. The dataset contains Z observations of the environment, $\mathcal{X}$, and the corresponding expert output trajectory, defined by a set of 2D waypoints in BEV space, i.e., $\mathcal{W} = \{\mathbf{w}_t = (x_t, y_t)\}_{t=1}^{T}$. This BEV space uses the coordinate frame of the ego vehicle. The policy for PlanT, π, is trained in a supervised manner using the collected data, $\mathcal{D}$, to minimize the loss function $\mathcal{L}$:

$$\underset{\pi}{\operatorname{argmin}} \; \mathbb{E}_{(\mathcal{X},\mathcal{W})\sim\mathcal{D}} \left[\mathcal{L}(\mathcal{W}, \pi(\mathcal{X}))\right]. \tag{1}$$

We use the L_1 distance between the predicted trajectory, $\pi(\mathcal{X})$, and the expert trajectory, $\mathcal{W}$, as the primary loss function, as well as an auxiliary objective based on future trajectory prediction which is detailed in Section 3 of the main paper.

1.2 Architecture

We first provide PyTorch-style pseudo-code for PlanT in Algorithm 1. In the following, we elaborate on the architecture of our waypoint decoder and low-level controller.

Decoder. We use a GRU decoder [2] to output waypoints. The update gate of the GRU takes the traffic light state and the current position as inputs. The input to the first GRU unit is set to (0,0) since the BEV space is centered at the ego vehicle's position. Identically to [1] we append the target point to the input of the GRU unit. We use a single layer GRU followed by a linear layer which takes in the hidden state and predicts the differential ego vehicle waypoints $\{\delta\mathbf{w}_w\}_{w=1}^{W}$ for $W = 4$ future time-steps in the ego vehicle's current coordinate frame. The waypoints are then given by $\{\mathbf{w}_w = \mathbf{w}_{w-1} + \delta\mathbf{w}_w\}_{w=t+1}^{t+W}$.

Controller. Our experimental setting assumes access to an inverse dynamics model [3] for low-level control, i.e., generating steer, throttle, and brake values provided by the future trajectory $\mathcal{W}$. Specifically, we use two PID controllers for lateral and longitudinal control from the predicted waypoints, $\{\mathbf{w}_w\}_{w=t+1}^{t+W}$. The longitudinal controller takes in the magnitude of a weighted average of the vectors

Algorithm 1 Planning Transformer pseudo-code (PyTorch style)

```
# V, R: attributes for vehicles and route segments (z, x, y, phi, w, h)
# W: ground truth waypoints
# e_cls, e_vehicle, e_route: learnable embeddings
# proj: linear projection layer
# transformer: transformer with global pairwise attention (BERT)
# gru: recurrent waypoint prediction module

# auxiliary task
# P: ground truth class labels for attributes of next time-step
# pred: linear classification layer per attribute

def PlanT(V, R):
    # compute token embeddings for vehicles and route
    vehicle_embed = proj(V) + e_vehicle
    route_embed = proj(R) + e_route

    # stack tokens as (CLS, v_1, ..., v_V, r_1, ..., r_R)
    input_embeds = stack(e_cls, vehicle_embed, route_embed)

    # use transformer to get hidden states
    hidden_states = transformer(input_embeds=input_embeds)

    # predict waypoints from hidden state for cls token
    W_preds = gru(unstack(hidden_states)[0])

    # predict attributes from hidden states for vehicle tokens
    v_hidden = unstack(hidden_states)[1:V]
    P_preds = []
    for attribute in ['z','x','y','phi','w','h']:
        P_preds.append(pred[attribute](v_hidden))

    return W_preds, P_preds

for (V, R, W, P) in dataloader:
    W_preds, P_preds = PlanT(V, R)
    loss = abs(w_preds - w)  # L1 loss
    for attribute, pred in enumerate(P_preds):
        loss += cross_entropy(pred, P[attribute]) # CE loss
    optimizer.zero_grad(); loss.backward(); optimizer.step()
```

between waypoints of consecutive time steps, whereas the lateral controller takes in their orientation. For the PID controllers, we use the same configuration as in the author-provided codebase of Chitta et al. [1].

1.3 Representation

Route Visualizations. We represent the route using a discrete set of tokens, each of which is a rectangular segment of the future route of the ego vehicle. To obtain these route segments, we subsample the dense set of points along the route provided by CARLA using the Ramer-Douglas-Peucker algorithm [4, 5].One segment spans the area between two of these subsampled route points, with the length equal to the distance between the points, as illustrated in Fig. 1 (green boxes). While we show all the segments with their original length in Fig. 1 for clarity, in practice, we restrict the maximum length of any single segment to L_{max} =10 m and always input a fixed number of segments $N_s = 2$ to our policy. We observe that increasing the value of N_s does not have a significant impact, but the performance drops without the restriction of L_{max} (Table 3b).

Object Extraction. We extract the object bounding boxes directly from CARLA. The boxes are then normalized to the coordinate frame of the ego vehicle before being input to PlanT. Note that the ego vehicle, which has a fixed position, orientation and extent in the chosen coordinate system, is excluded from the representation. The effect of including the ego vehicle token, which provides information regarding the ego vehicle's velocity, is minimal (Table 3c).

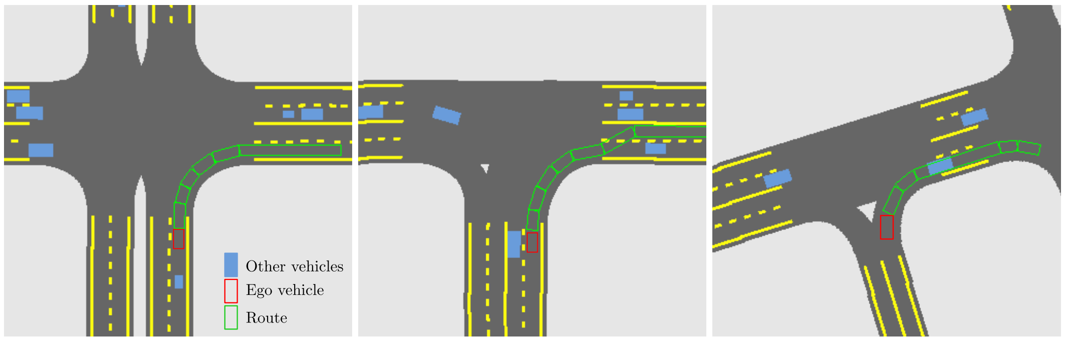

Figure 1: **Route Representation.** We visualize example route segments used to obtain the object-level input representations for PlanT that encode the road context as green oriented bounding boxes.

1.4 Metrics

In the main paper, we report five metrics to provide a comprehensive understanding of the performance of our planners. We now provide detailed descriptions of these metrics.

(1) **Route Completion (RC)**: percentage of route distance completed, R_i by the agent in route i, averaged across N routes:

$$\text{RC} = \frac{1}{N}\sum_{i}^{N} R_i. \tag{2}$$

However, if an agent drives outside the route lanes for a percentage of the route, then the RC is reduced by a multiplier (1- % off-route distance).

(2) **Infraction Score (IS)**: product of infraction penalty coefficients, p^j for every instance of infraction j incurred by the agent during the route:

$$\text{IS} = \prod_{j}^{\text{Ped,Veh,Stat,Red}} (p^j)^{\#\text{ infractions}^j}. \tag{3}$$

Agents start with a perfect 1.0 base score, which is reduced by a penalty coefficient for every infraction. The penalty coefficient for each infraction is pre-defined and set to 0.5 for a collision with a pedestrian, 0.6 for a collision with a vehicle, 0.65 for a collision with static layout elements, and 0.7 for red light violations. The official CARLA leaderboard also mentions a penalty for stop sign violations. However, we omit this infraction from our analysis as it is known to have no impact on the official leaderboard [1].

(3) **Driving Score (DS)**: weighted average of the route completion with infraction multiplier P_i:

$$\text{DS} = \frac{1}{N}\sum_{i}^{N} R_i P_i. \tag{4}$$

(4) **Collisions with Vehicles per km (CV)**: total number of vehicle collisions, normalized by the total number of km driven:

$$\text{CV} = \frac{\sum_i^N \#\text{ collisions}_i}{\sum_i^N k_i}, \tag{5}$$

where k_i is the driven distance (in km) for route i.

(5) **Inference time (IT)**: time in milliseconds for one forward pass of the learned planning model during inference with a batch size of 1. All inference times are recorded on a single RTX 3080 GPU. An IT below 33ms is sufficient for real-time execution (30fps) of the planner on this hardware.

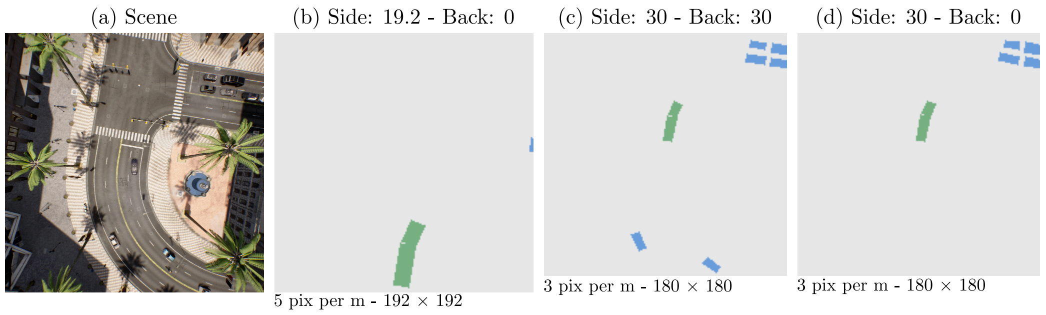

Figure 2: **PlanCNN Input Rasterization.** We consider 3 different settings for spatial extent and resolution in the ablation study for PlanCNN (Table 2a of main paper). Here, we visualize the vehicle channel (in blue) and ego route channel (in green) for the different configurations.

2 Implementation Details

In this section, we first describe two baselines, rule-based planner and PlanCNN, in greater detail. We then elaborate on the model configurations for PlanT that we use in our experiments. Finally, we describe the implementation of the perception module used for PlanT with perception.

2.1 Rule-based Planner

We build a rule-based planning algorithm that uses the same inputs as PlanT based on the expert driver from [6]. Given the route to be followed, this planner chooses the nearest point along the route 4m from its current position to compute an aim direction, which is fed to a lateral PID controller to output steering values. For longitudinal control, the planner uses either 0 or 4 m/s as the target speed depending on its current circumstances. If it is less than 5 meters away from a pedestrian, vehicle, or red light, the target speed is set to 0 m/s. For vehicle collisions, we additionally check if a linear extrapolation of the ego vehicle and target vehicle based on their current velocity will lead to them being closer than 5 meters after 4 seconds, in which case the target speed is set to 0. Under every other circumstance, the target speed is set to 4 m/s. The parameter choices for these heuristics were made by tuning for the best DS on a subset of the Longest6 benchmark. A longitudinal PID controller then tries to match the speed of the expert policy to this target speed as much as possible. We use the same hyper-parameters for the PID controllers as those in Prakash et al. [6].

2.2 PlanCNN

This approach is inspired by the AIM-BEV model of Hanselmann et al. [7] and AIM-VA model of Chitta et al. [8]. PlanCNN includes 3 channels for (1) vehicles, (2) vehicle speeds, and (3) the ego route, and is the closest representation to the one used by PlanT (i.e., the same input information as PlanT but rasterized into an image). The vehicle and ego route channels have binary values indicating the presence/absence of that class at a particular pixel location. The vehicle speed channel encodes the speed of each pixel in the scene (in meters per second) and is set to zero for all pixels that do not belong to the vehicle class.

In Table 2a of the main paper, we explore three different ways of rasterizing the inputs for PlanCNN. We provide illustrations of these approaches to rasterization in Fig. 2. More specifically, we visualize the vehicle channel and ego route channel for the driving scene shown on the left (Fig. 2a). The first configuration (Side 19.2 and Back 0, Fig. 2b) is taken from existing work [9, 7]. In this representation, the ego vehicle is placed at the bottom of the image. The spatial region covered in the image includes $38.4\,\mathrm{m}$ to the front of the ego vehicle and $19.2\,\mathrm{m}$ to either side at a resolution of 5 pixels per meter, resulting in a 192×192 image. The second configuration (Side 30 and Back 30, Fig. 2c) is closest to the input used by PlanT. It uses a 180×180 image with a resolution of 3 pixels per meter. As an intermediate step between these two configurations, we additionally consider a version of PlanCNN with the same spatial extent as PlanT to the sides but with no visibility of vehicles to the rear. For this, we do not render any pixels on the bottom half of the image, as shown in Fig. 2d. We show that the configuration closest to that of PlanT (Fig. 2c) obtains the best results (see Table 2a of the main paper), indicating the importance of a larger field of view.

Model	Layers	Hidden size	Heads	Parameters	Inference Time
PlanT$_{\texttt{MINI}}$	4	256	4	11.2M	5.46$\pm$0.28
PlanT$_{\texttt{SMALL}}$	4	512	8	28.8M	7.04$\pm$0.39
PlanT$_{\texttt{MEDIUM}}$	8	512	8	41.4M	10.79$\pm$0.47

Table 1: **PlanT Variants.** Details regarding the size and runtime of our model variants.

Note that we use the same waypoint Prediction Network (i.e., GRU) and controller for the feature vector obtained by the CNN encoder of PlanCNN and PlanT, enabling fair comparisons.

2.3 PlanT

Our analysis includes three BERT encoder variants taken from [10]: `MINI`, `SMALL`, and `MEDIUM` with 11.2M, 28.8M and 41.4M parameters respectively. We provided detailed information regarding these architectures in Table 1. The transition from PlanT$_{\texttt{MINI}}$ to PlanT$_{\texttt{SMALL}}$ involves an increased width (hidden dimension size), whereas the transition from PlanT$_{\texttt{SMALL}}$ to PlanT$_{\texttt{MEDIUM}}$ increases the network depth (number of layers).

2.4 PlanT with Perception

We leverage a TransFuser [1] backbone for LiDAR and RGB sensor fusion in order to predict the input attributes for PlanT. Unless otherwise stated, our implementation details follow [1].

Sensor configuration. We use LiDAR and RGB sensors to extract the input representation for PlanT from sensor data. More specifically, we use the front half of a LiDAR that provides points in the range $32\,\mathrm{m}$ to the front and $16\,\mathrm{m}$ to each side of the vehicle, converted to a 2D BEV grid as described by [1]. For the RGB sensor we use the configuration of [11], namely one camera to the front placed $1.5\,\mathrm{m}$ behind and $2\,\mathrm{m}$ above the ego vehicles origin with a resolution of 900×256 and a field of view of $100°$.

Training. We keep the architectural details but adjust some of the training objectives of TransFuser to obtain a suitable perception module and add data augmentation. Our training objectives include the original vehicle detection loss of TransFuser based on a CenterNet [12] detection head and a new waypoint objective supervised with the next 20 dense ground truth route waypoints instead of the future waypoints of the ego vehicle. We also learn the initial position input to the GRU since the route can be displaced and not start at the origin of the vehicle when the ego vehicle deviates from the route. Finally, we add a binary traffic light classification flag. For data augmentation we shift and rotate the position of the camera as a proxy for deviations of the ego vehicle during data collection. We uniformly sample the augmentation values from [-1, 1] meters for the shift and [$-5°$, $5°$] for the orientation. The positions and orientations of the other vehicles and the lidar points are adjusted accordingly in a pre-processing step of the training.

Inference. To be able to input the obtained information to PlanT we post-process the output of the perception module to match the input representation of PlanT. For the vehicles, we only consider detections with a confidence above a threshold ϕ taken from the CenterNet output heatmap. The perception network predicts bounding boxes for every time step but does not associate instances of the same entity, which is required for extracting vehicle speeds for the next stage of our system. To match instances at runtime, we store a buffer of bounding boxes of the last ξ timesteps sampled at $20\,\mathrm{fps}$ and associate them using Hungarian matching [13] based on the intersection over union. The matching problem is relatively simple in the BEV coordinate system with this large frame rate since objects do not occlude each other. Objects that do not match with detections from the last ξ timesteps are excluded from the state representation for PlanT.

To obtain the route segments we accumulate route predictions over time to get a dense set of points along the route. We then fit a cubic spline to these points, making it possible to interpolate smoothly through them and remove outliers. We then resample equidistant points along this spline, leading to an input $\mathcal{U}_t$ similar to what the route planner of the CARLA simulator provides. To obtain route segments, we use the same extraction procedure described in the main paper. The points are subsam-

Loss weight λ	DS $\uparrow$	RC $\uparrow$	IS $\uparrow$	CV $\downarrow$
0.6	73.86$\pm$0.89	86.48$\pm$1.57	0.85$\pm$0.02	0.30$\pm$0.07
0.8	72.74$\pm$3.58	86.76$\pm$3.30	0.83$\pm$0.02	0.37$\pm$0.04
1.0	76.31$\pm$3.13	87.54$\pm$2.67	0.86$\pm$0.02	0.29$\pm$0.05
2.0	67.83$\pm$3.79	86.74$\pm$2.85	0.77$\pm$0.03	0.49$\pm$0.08
4.0	67.35$\pm$4.69	84.94$\pm$4.01	0.79$\pm$0.03	0.44$\pm$0.13

Table 2: **PlanCNN Multi-task Training.** Mean$\pm$std of 3 evaluations on Longest6. None of the loss weight settings lead to improvements compared to training PlanCNN with only the imitation objective (Table 1 main paper).

pled using the Ramer-Douglas-Peucker algorithm [4, 5] to select a subset $\hat{\mathcal{U}}_t$. One segment spans the area between two points subsampled from the route, $\mathbf{u}_{i,t}, \mathbf{u}_{i+1,t} \in \hat{\mathcal{U}}_t$.

3 Additional Experiments

In this section, we provide additional results. We first show the impact of different training strategies involving auxiliary objectives for PlanCNN and PlanT. Next, we provide additional insights into the design choices for the input representation of PlanT. Finally, we comment on the training variance, failure cases of the expert, and our attempt to extract attention maps for PlanCNN via GradCAM [14].

3.1 Training Strategies

Multi-task Training for PlanCNN. The default configuration of PlanCNN (see Table 1 main paper) uses only the imitation objective with no auxiliary training loss. Here, we investigate multi-task training for PlanCNN via a convolutional decoder. The self-supervised multi-task loss function used for PlanCNN in this experiment can be described as:

$$\mathcal{L} = \sum_{t=1}^{T} ||\mathbf{w}_t - \mathbf{w}_t^{gt}||_1 + \lambda \, ||\hat{\mathbf{s}}_{t+1} - \mathbf{s}_{t+1}||_1, \tag{6}$$

where $\mathbf{w}_t$ represents the waypoint at time-step t, and $\mathbf{w}_t^{gt}$ represents the corresponding ground truth. $\hat{\mathbf{s}}_{t+1} \in \mathbb{R}^{180\times180\times3}$ is the predicted rasterized input representation of the next time-step, with $\mathbf{s}_{t+1}$ being the corresponding ground truth. We use the feature map after the fourth ResNet layer before average pooling and process it with a convolutional decoder to predict the BEV of the next timestep. We report the performance of this multi-task variant of PlanCNN with different settings of the loss weight λ in Table 2. We performed a hyperparameter search in $[0.6, 4.0]$ which ensured that the magnitude of the auxiliary loss was in a similar range to the imitation objective. This is similar to how the weighting factor λ for PlanT (cf. Eq. (1) in the main paper) was determined. None of these settings lead to significant improvements when compared to the default version from the main paper trained with only the imitation objective (which reaches a DS of 77.47).

PlanT Training Strategies. We show results for three different training strategies for PlanT$_{\texttt{SMALL}}$ in Figure 3. Specifically, we demonstrate that using the auxiliary task of predicting the attributes of other vehicles in a future timestep significantly improves performance and enables the use of small datasets. For this experiment, we randomly subsample 50% of the dataset (0.5×) in addition to the three versions of the dataset used in the main paper (1×, 2×, 3×). We compare three different training techniques: (1) Pure imitation learning without an auxiliary task. (2) Pre-training, where we train the model in two stages. First, we pre-train on the auxiliary task of predicting the other vehicle's future. Note that we only do this for the vehicle tokens, so any common driving dataset can be used for this step. After pre-training the network weights we fine-tune the model on the waypoint prediction task with only the imitation objective. (3) Multi-task learning on both objectives simultaneously. On all dataset sizes, we observe that pre-training helps to improve the performance compared to imitation only, but is not able to reach the performance of the multi-task approach. In particular, we observe that on the 0.5× dataset, multi-task training improves the DS by over 30

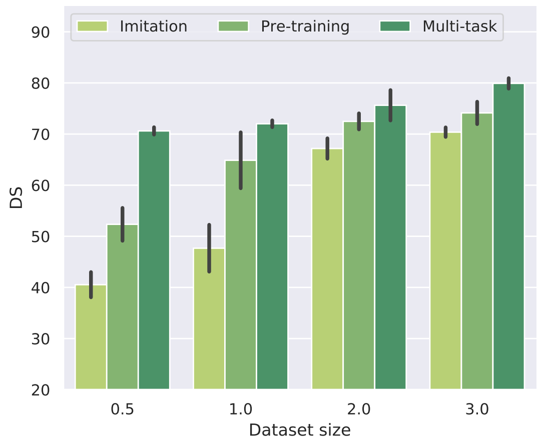

Figure 3: **Training Strategy.** Driving scores on the Longest6 benchmark for different dataset sizes. Evaluation std (3 runs) is shown using black error bars. With our proposed multi-task training strategy we gain 30% DS for PlanT$_{\texttt{SMALL}}$ on the 0.5× dataset and reach the same performance obtained with the 3× dataset (which is six times larger) when trained using only the imitation objective.

λ	DS ↑
0.05	59.84±1.97
0.1	70.15±5.76
0.2	72.01±0.82
0.4	68.63±2.03

(a) **Loss weights.**

Route	DS ↑
no L_{max}	62.78±2.82
2 segments	72.01±0.82
4 segments	73.01±3.41
6 segments	70.63±4.38

(b) **Route**.

v_i	DS ↑
- speed	49.07±3.16
- id	67.32±5.54
default	72.01±0.82
+ ego	72.28±5.82

(c) **Attributes**.

Tok.	Enc.	DS ↑
Obj.	O	72.01±0.82
Obj.	MLP	71.15±6.69
Attr.	A+O	60.57±4.28
Attr.	A+P	47.69±2.22

(d) **Tokenization**.

Table 3: **PlanT Ablation Experiments.** We investigate different parts of our method including the loss weights for the multi-task setting, different versions of the input representation and how to obtain token embeddings. To reduce the computational overhead, the ablation studies are conducted using PlanT$_{\texttt{SMALL}}$ on the 1× dataset. The score of the default configuration is highlighted in gray.

points compared to imitation only. By doing so, it reaches the performance of the imitation-only model trained with a much larger dataset (e.g. 3×). In summary, multi-task training, while being simpler and faster than pre-training since it does not involve two stages, is the best training strategy for PlanT.

PlanT Loss Weights. In Table 3a, we compare the DS of PlanT for different settings of the auxiliary loss weight λ, with a weight of 0.2 giving the best results. Therefore, we use this value in all the experiments involving PlanT in both the main paper and this supplementary document.

3.2 PlanT Input Representation Hyperparameters

Route. Table 3b shows the impact of changing the route representation on the driving score. In the default setting, we reduce the maximum length of one route segment to $10\,\mathrm{m}$ and include two tokens for the two closest route elements in PlanT's input representation. Intuitively, the maximum length limit reduces out-of-distribution data samples during test time since long straight routes (that can span several hundreds of meters with a single bounding box) are represented in the same way as short ones. This prevents sizes unseen during training from occurring during testing and makes it easier to generalize to unseen road layouts. We also observe this empirically (Table 3b), where not limiting the maximum length hurts performance. Using more than 2 route segments as an input does not have a significant impact, but with more than 4 segments the performance decreases slightly. Therefore, for simplicity and better inference times, we use 2 route segments by default.

Attributes. We study the design choices involved in the input attributes in Table 3c. Removing the speed attribute from the vehicle input tokens (- speed) has a large negative impact, showing that the motion is an important cue for high driving performance. Removing the route id (- id) from the input (without the id the ordering of the route segments is not explicitly known) also slightly hurts

ϕ	DS $\uparrow$	CV $\downarrow$
0.1	57.66$\pm$5.01	**0.97**$\pm$0.09
0.3	56.03$\pm$1.08	1.18$\pm$0.21
0.5	48.75$\pm$2.54	1.70$\pm$0.12

(a) **CenterNet Threshold**.

ξ	DS $\uparrow$
2	55.34$\pm$6.71
4	57.66$\pm$5.01
6	52.63$\pm$7.66

(b) **Matching Threshold**.

Table 4: **PlanT with Perception Ablation Experiments.** We investigate different approaches to post-process the perception outputs, specifically the confidence threshold of the vehicle predictions and the matching threshold. The score of the default configuration is highlighted in gray.

performance. Since we only use the closest two route segments, not knowing the order does not hurt as much. Interestingly, unlike previous work [1], we find that inputting the ego velocity (+ ego) does not hurt performance. As we mention in the main paper, PlanT also does not suffer from the *inertia problem* [15], and creeping or other heuristics to prevent this are necessary.

Tokenization. We attempted several different methods to tokenize our input representation (Table 3d). Our default setting encodes one token per object and adds an object-specific learnable embedding (O). When we replace the linear projection and the addition of the object-specific embedding with an object-specific MLP (Enc. denotes the MLP), we reach a similar performance. For the experiments in the main paper, we choose the linear projection instead of an MLP since a linear projection is a more standard approach for transformers [16]. We also experiment with two versions of PlanT with one token per attribute (Tok.=Attr.) In this configuration, we use attribute-specific learnable embeddings (A) in addition to the object-specific embedding (O). This configuration (Attr.+A+O) is around 10 points worse (DS=60.57) than the default approach of one token per object (DS=72.01). Since we have 6 attributes, using one token per object also leads to a 6$\times$ reduction in the input length to the transformer, saving computation due to its quadratic complexity. Finally, we attempt to use one token per attribute with knowledge about the attribute type (A) and a sinusoidal (non-learnable) positional embedding (P). This decreases the performance further to 47.69, due to the lack of explicit information about the object type, which the transformer is unable to recover.

3.3 PlanT with perception

Failures. The main failure mode that arises due to the incorporation of a perception module is a 3$\times$ increase in CV, resulting in a significant drop in performance for PlanT with perception (DS=57.66) in comparison to PlanT with perfect perception (DS=81.36). This is a result of two main factors: (1) the incomplete field of view which excludes the back and far away objects to the sides, which we know are important based on Tab. 2a of the main paper and (2) the added input noise due to inaccuracies in the perception system.

Ablation study. We first evaluate the influence of the CenterNet confidence threshold ϕ in Table 4a. Reducing the threshold results in keeping more vehicle detections that have a lower confidence to be a correct prediction. This leads to more detected vehicles and is reflected in the decrease of the collision rate CV and therefore also in the improvement of the driving score DS. Varying the threshold for the matching algorithm shows that 4 leads to the best result, as can be seen in Table 4b.

Runtime of the full driving stack. For calculating the runtime of the full driving stack the inference time of the object detector needs to be added to the inference time of the planner. When considering real-time requirements of the full driving stack (around 30 fps, i.e., 33 ms), decreasing the inference time of the planner to 5.46 ms compared to 28.94 ms without any loss in performance would permit the use of a significantly more accurate object detector while maintaining the required system runtime. With many real-world detectors, the number of input actors may increase in environments where the perception task is more challenging. To evaluate the impact of such an increase, we scale the number of input vehicle tokens to 2x the true number of vehicles in every frame and observe that the planner's runtime increases by only 9.6%. Therefore, our conclusions regarding the improvements in inference time for PlanT compared to PlanCNN remain true in a setting involving false positive detections. For the specific perception module we used in our experiment we obtain a runtime of $26.82\,\mathrm{ms}$ (including the forward pass of the model, matching of the vehicles, interpolation and subsampling of the route), leading to a total inference time of $37.61\,\mathrm{ms}$ $(26.82 + 10.79)$.

	Method	Input	DS ↑	RC ↑	IS ↑	CV ↓	IT ↓
Sensor	LAV [17]	Camera + LiDAR	32.74±1.45	70.36±3.14	0.51±0.02	0.84±0.11	-
	TransFuser [1]	Camera + LiDAR	47.30±5.72	**93.38±1.20**	0.50±0.60	2.44±0.64	101.24
	LAV v2 [17]	Camera + LiDAR	57.52±1.16	82.61±0.99	**0.68±0.02**	**0.69±0.08**	-
	PlanT w/ perception	Camera + LiDAR	**57.66±5.01**	88.20±0.94	0.65±0.06	0.97±0.09	37.61
	Expert [1]	*Obj. + Route + Actions*	*76.91±2.23*	*88.67±0.56*	*0.86±0.03*	*0.28±0.06*	-

Table 5: **Longest6 Results.** We show the mean±std for 3 evaluations. In addition to the main paper we also evaluate LAV v2 which was made public after the submission.

Model	Mean DS ↑	Train std ↓	Eval std ↓
PlanCNN	74.94	2.91	2.07
PlanT$_{\texttt{MEDIUM}}$	77.90	3.19	3.00

Table 6: **Variance.** We show training and evaluation variance for 3 training seeds of PlanCNN and PlanT$_{\texttt{MEDIUM}}$ trained on the 3× dataset, with 3 evaluation runs each. Both models exhibit large variance in scores, both in terms of training and evaluation.

3.4 Other Results

LAV v2. We evaluate the LAV v2 [17] checkpoint that was released after the submission deadline on the Longest6 benchmark. In Tab. 5 we observe that this model can match but does not outperform PlanT with perception.

Variance. We show the impact of training and evaluation seed variance in Table 6. We train each method 3 times and evaluate each resulting model 3 times on the Longest6 benchmark, giving a total of 9 evaluations per method. The training variance between different seeds is due to different data augmentations, data sampling, and network initializations during optimization. The evaluation variance is a result of the variation in the sensor noise, physics and traffic manager of CARLA. Based on the results, both factors are considerable.

Expert Failures. We show examples of common failure modes of the expert in Fig. 4. These infractions all occur inside intersections after the expert initially enters since it does not anticipate the entrance of other traffic at the same time. Once the other vehicle enters, the expert predicts a collision and stops. In such situations, the expert remains static, and there is a chance of getting blocked (when the other vehicle also stops and remains static) or being involved in a collision (when the other vehicle ignores the expert due to CARLA's imperfect rule-based traffic manager). We observe that PlanT is more conservative when deciding whether to enter an intersection and encounters such situations slightly less often.

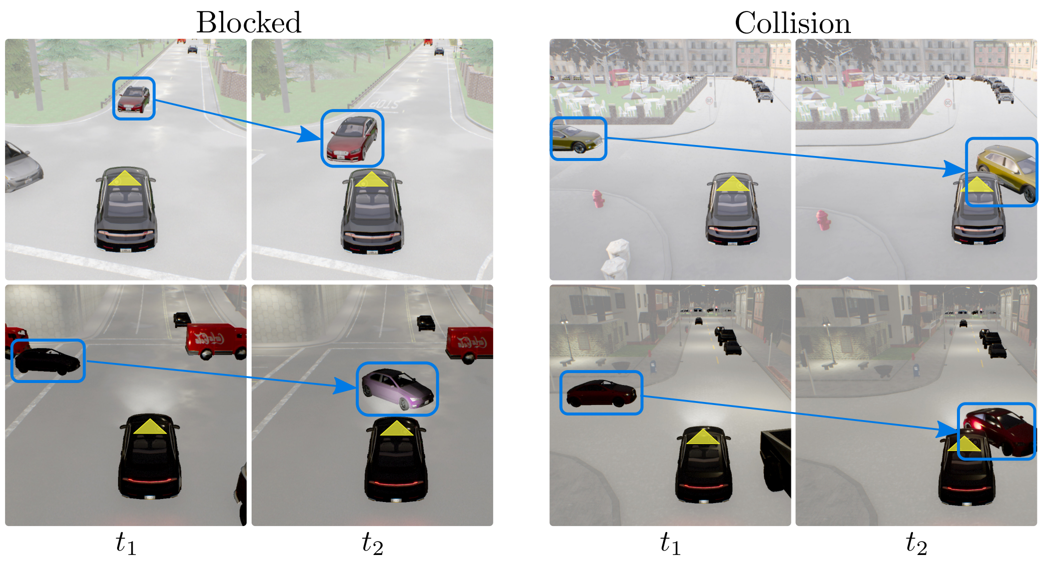

Figure 4: **Expert Failures.** The expert's failures are a result of stopping after entering an intersection due to anticipated collisions. The ego vehicle is marked with a yellow triangle, and vehicles that lead to infractions are marked with a blue box.

References

[1] K. Chitta, A. Prakash, B. Jaeger, Z. Yu, K. Renz, , and A. Geiger. Transfuser: Imitation with transformer-based sensor fusion for autonomous driving. *IEEE Trans. on Pattern Analysis and Machine Intelligence (PAMI)*, 2022.

[2] K. Cho, B. van Merrienboer, Ç. Gülçehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Proc. of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014.

[3] R. Bellman. *Adaptive Control Processes - A Guided Tour*, volume 2045. Princeton University Press, 2015.

[4] U. Ramer. An iterative procedure for the polygonal approximation of plane curves. *Comput. Graph. Image Process.*, 1:244–256, 1972.

[5] D. H. Douglas and T. K. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 10:112–122, 1973.

[6] A. Prakash, K. Chitta, and A. Geiger. Multi-modal fusion transformer for end-to-end autonomous driving. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[7] N. Hanselmann, K. Renz, K. Chitta, A. Bhattacharyya, and A. Geiger. King: Generating safety-critical driving scenarios for robust imitation via kinematics gradients. *arXiv.org*, 2204.13683, 2022.

[8] K. Chitta, A. Prakash, and A. Geiger. Neat: Neural attention fields for end-to-end autonomous driving. In *Proc. of the IEEE International Conf. on Computer Vision (ICCV)*, 2021.

[9] D. Chen, B. Zhou, V. Koltun, and P. Krähenbühl. Learning by cheating. In *Proc. Conf. on Robot Learning (CoRL)*, 2019.

[10] I. Turc, M.-W. Chang, K. Lee, and K. Toutanova. Well-read students learn better: On the importance of pre-training compact models. *arXiv.org*, 1908.08962, 2019.

[11] P. Wu, X. Jia, L. Chen, J. Yan, H. Li, and Y. Qiao. Trajectory-guided control prediction for end-to-end autonomous driving: A simple yet strong baseline. *arXiv preprint arXiv:2206.08129*, 2022.

[12] X. Zhou, D. Wang, and P. Krähenbühl. Objects as points. *arXiv.org*, 1904.07850, 2019.

[13] H. W. Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2:83–97, 1955.

[14] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proc. of the IEEE International Conf. on Computer Vision (ICCV)*, pages 618–626, 2017.

[15] F. Codevilla, E. Santana, A. M. López, and A. Gaidon. Exploring the limitations of behavior cloning for autonomous driving. In *Proc. of the IEEE International Conf. on Computer Vision (ICCV)*, 2019.

[16] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *Proc. of the International Conf. on Learning Representations (ICLR)*, 2021.

[17] D. Chen and P. Krähenbühl. Learning from all vehicles. In *CVPR*, 2022.