Supplementary Material for Hidden Biases of End-to-End Driving Models

Bernhard Jaeger Kashyap Chitta Andreas Geiger
University of Tübingen Tübingen AI Center
{bernhard.jaeger, kashyap.chitta, a.geiger}@uni-tuebingen.de

Abstract

*This **supplementary document** provides detailed implementation details of the reproduced TransFuser and TransFuser++ methods. It describes the dataset and labeling algorithm used in the study. Finally, we provide additional experimental results and qualitative examples. Qualitative results are also visualized in the **supplementary video.***

1. Changes to TransFuser

Our reproduced TransFuser largely follows [7] but has some minor differences in implementation details described in this section.

1.1. Expert

For data collection, we follow the common practice of using an automatic labeling algorithm (expert driver) to generate the imitation labels. The auxiliary perception labels are provided directly by the CARLA simulator except for the bird's eye view (BEV) segmentation, for which we follow [25] and render the relevant objects into an HD map. To generate the imitation labels, the expert driver needs to solve planning and control, but can bypass perception using privileged access to the simulator. We build our expert upon the model predictive control (MPC) approach of [7]. Lateral control is done by following the next point (at least 3.5 meters away) in a path, created by an A* algorithm, with a PID controller. Longitudinal control is done via MPC that differentiates between 4 target speeds. For regular driving, we use 8 m/s (double the target speed compared to [7]). Inside intersections, we slow down to 5 m/s.

Collision avoidance: The target speed is set to 0 m/s when the MPC algorithm predicts a collision. We predict collisions similarly to [7]. They approximate all agents' future positions by iteratively unrolling a kinematic bicycle model [4]. Actions for other cars are set to be the same as the current time step while the ego agent's own action is approximated by using a PID controller to follow the A* path. In case the ego agent's bounding box overlaps with another agent's bounding box at future time step t, a collision is predicted. To keep a safety distance to the leading vehicle, the MPC expert in [4] additionally has a static bounding box in front of the ego agent that predicts a collision if it intersects with any other agent. Because our agent has double the maximum driving speed, we need to keep a larger safety distance. Using a static area would not suffice here, as the static bounding box will become so large that it sticks into the opposing lane during turns, causing unnecessary braking. Instead, we approximate the area where the expert would end up if it performed a full brake (after 1 meter of driving) at his current speed s. The distance to stop d is approximated with:

$$d = 0.5 * (\frac{s * 3.6}{10.0})^2 + 2.5 \quad (1)$$

We again use the kinematic bicycle model to compute where the expert will be after this distance. If the bounding box of the expert at that time step intersects with any bounding boxes for the current time step, we set the target speed to 0. This approach has the advantage that the safety area follows the road and increases with increasing agent speed. A second problem coming from the higher driving speed is scenario 3 where a pedestrian runs in front of the vehicle. The expert will stop as soon as the pedestrian attempts to move forward, but when driving at 8 m/s this is already too late to prevent the collision. To solve this problem, we preemptively slow down to 2 m/s whenever a pedestrian is within a 30-meter radius in front of the vehicle.

Traffic rules: Stop signs infractions are addressed by slowing down to 2 m/s when the safety area intersects with the stop sign trigger area (provided by the simulator) and stopping once the car is on the stop sign trigger area. Red lights are resolved by setting the target speed to 0 when the safety area or vehicle intersects with the entrance of an intersection and the corresponding traffic light is active.

Performance: We show in Table 1 that our expert outperforms the baseline by 4 DS. This represents a higher upper bound for our imitation learning models. However, since

(a) TransFuser (NC conditioned) (b) TransFuser (TP conditioned) (c) TCP [23] (TP conditioned)

Figure 1: **Extrapolation to target point.** Additional example of an unknown situation where TP conditioned methods extrapolate their waypoints towards target points. The TP conditioned models succeed to drive back to the lane, while the discrete conditioned model gets stuck on the sidewalk.

Method	DS ↑	RC ↑	Veh ↓	Block ↓
MPC Expert [7]	77 ± 2	89 ± 1	0.28	0.13
MPC Expert (ours)	**81** ± 3	**90** ± 1	**0.21**	**0.09**

Table 1: **Expert comparison on Longest6**

no method can currently reach that upper bound, we see no direct improvement from the better expert (see Table 2).

1.2. Dataset

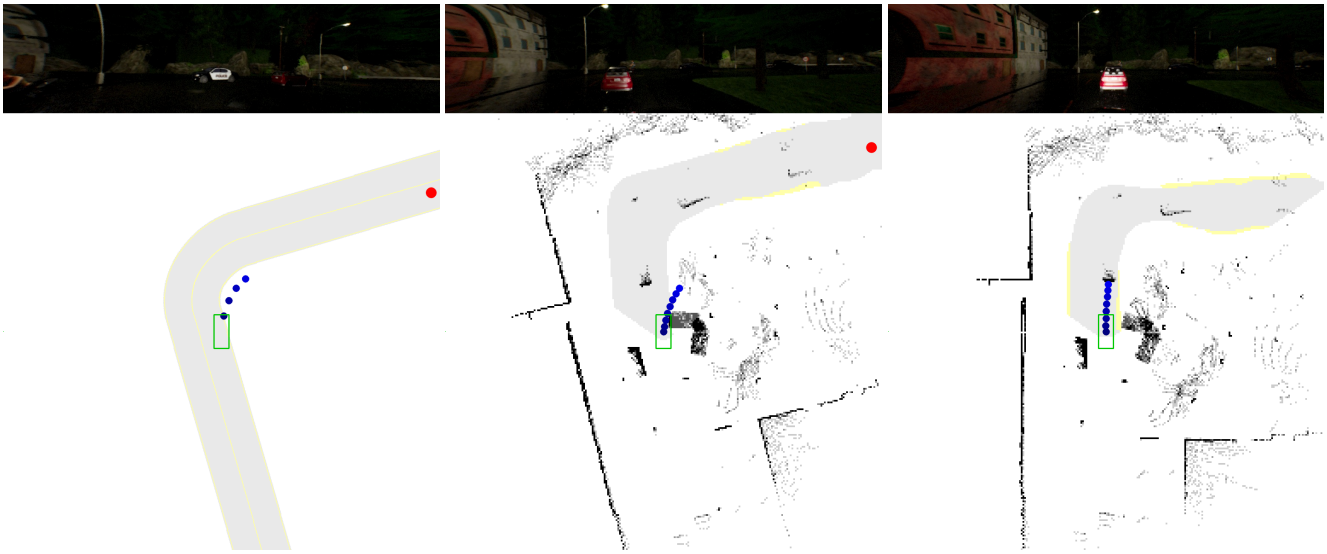
We generate our training dataset by executing the expert described in Section 1.1 on the training routes from [7] and storing every frame (20 FPS). Weathers are randomized per route instead of per frame to avoid exposure problems. During training, we train on every fifth frame, leading to an effective FPS of 4. For the scaling experiment, we rerun data collection on each route 3 times. This randomizes weathers and traffic (the traffic manager in CARLA 0.9.10 is not deterministic) but the environment is the same.

For the final model in the main paper we recollect the data at 4 FPS and train on every frame. The model sees equivalent data, but this makes the second dataset is 5x smaller and hence easier to release. To reduce storage requirements further, we compress the camera images with JPG (we add the compression during inference as well to avoid distribution shifts), perception labels with PNG (depth maps are stored at 8 bit resolution), text files with zip, and LiDAR point clouds with LASzip [10] (which compresses point clouds much better than standard zip). We store full 360° LiDAR sweeps and realign all points into the coordinate system of the current frame. Since the expert is not a perfect driver, we log the driving score of each training route and only train on routes with 100 DS. The labels for the disentangled path are generated by storing the next 10 2D points (spaced 1 meter apart) from the A* path the expert is following.

1.3. Training and Architecture

Training: We use the same loss functions as in [7] for all 5 outputs. For the new target speed classification task we use a class frequency weighted cross-entropy loss with label smoothing 0.1. As classes we use the 4 different target speeds of the expert (see Section 1.1). We observe some failure cases where the target speed predictions are overconfident, and the car starts braking a few frames too late when stopping behind another vehicle, leading to close rear collisions. Label smoothing addresses this problem because it makes the car drive a bit slower (due to the probability weighting). This gives the car a few more frames time to brake and avoids these collisions. To make this more explicit, we do not use label smoothing in the final model in the main paper, and instead reduce the target speeds by 2 m/s during inference, achieving the same effect.

During training, each individual loss is added together with the same weight. The loss weights are normalized to sum to 1. We use AdamW [13] with amsgrad [17] and a slightly higher learning rate of 0.0003 which is reduced by a factor of 10 after epoch 30. We train for 31 epochs and always use the last epoch as our model. All models in Sec-



(a) TCP [23] follows a shortcut. (b) TransFuser follows a shortcut. (c) A transformer decoder fixes this.

Figure 2: **Target point shortcut.** When TP conditioned methods extrapolate to spatially distant waypoints, they incur large steering errors. Replacing global average pooling in TransFuser with a cross-attention mechanism mitigates the issue.

tion 3 of the main paper are trained on 8 2080ti GPUs with distributed data parallel and total batch size of 48. For the dense dataset, we subsample by a factor of 5 but shift the first frame by the GPU index, so that all the frames are seen during training (a subtle form of data augmentation, close by frames are mostly redundant).

The final model is trained with 4 A100 GPUs and batch size 128 to speed up training. We also add standard color augmentations to the camera and a discrete conditioning concatenated with the velocity input, so that the target speed branch also has a conditioning signal. This is conceptually more sound, but we observed no significant impact on driving performance. Our path labels have a subtle ambiguity due to conversion from a path in global coordinates to the local ego coordinate system (points can be shifted by 1 meter depending where the car is on the path). To resolve this, we linearly interpolate between the stored points at 1 meter distances in ego coordinates during training. This makes the predictions look qualitatively more consistent, but again has no significant impact on driving score.

Architecture: For the BEV semantic segmentation we predict 11 classes (unlabeled, road, sidewalk, non cross-able lane marker, cross-able lane marker, stop signs, traffic light stop line green / yellow / red, vehicle, pedestrians) [25]. We only predict pixels that are visible in the camera, since some classes need RGB features. For the perspective segmentation we predict 7 classes (unlabeled, road, sidewalk, lane markings, traffic light, vehicle, pedestrian). For the bounding boxes we predict 4 classes (red and yellow traffic light stop line, stop sign, vehicles and pedestrians). Traffic lights and stop sign boxes are only predicted if they affect the ego vehicle. Vehicles and pedestrians only when they are in the LiDAR range. Architecture wise we add 1x1 convolution layers in the LiDAR branch before each transformer that match the channel dimension of the LiDAR to the one in the image branch (allows using different backbones for the two branches, though we keep using RegNetY-3.2GF [16] for both). Our reproduced TransFuser predicts 8 waypoints, each placed 250 ms apart (up to 2 seconds into the future).

Sensors: We follow [23] and use a single high resolution camera. It has a 110° horizontal field of view (FOV), is mounted at (-1.5, 0.0, 2.0) and has a resolution of 256 x 1024. Our LiDAR is mounted at (0.0,0.0, 2.5) and has a full 360° FOV. LiDAR points are realigned into the current vehicle coordinate system by using the filter described in Section 1.4. The LiDAR points are voxelized into a 256x256 grid representing a 64 x 64 meter area with the vehicle at its center. Each pixel covers a 0.25 x 0.25 meter area. We remove the LiDAR ground plane by removing all points with a height of less than 0.2 meters.

Performance: The reproduced TransFuser has a slightly lower DS to the original on the Longest6 benchmark as shown in Table 2. The difference in driving score likely is a result of the driving score weighting the dissimilar failure modes differently. Note, that our reproduced TransFuser is a single model, whereas the original result was from an ensemble of 3 models.

1.4. Localization

We localize our vehicle with a GNSS sensor. The GNSS signal has Gaussian noise applied to it, leading to average localization errors of $\sim$0.7 meters. Like prior work [5, 7]

Method	DS ↑	RC ↑	Veh ↓	Stat ↓
TransFuser (ours)	40	82	**1.17**	0.57
TransFuser [7]	**47**	**93**	2.45	**0.07**

Table 2: **Reproduced TF vs original on Longest6**

we use a filtering algorithm to reduce the noise. In particular, we use an Unscented Kalman Filter (UKF) [19,22] with Van der Merwe's scaled sigma point algorithm [14]. As its model of the car, our UKF uses the same kinematic bicycle model as the expert [4]. The filter tracks the position, orientation and speed of the ego vehicle. The parameters of the filter are tuned manually by reducing localization error on a small dataset consisting of ground truth localization paired with GNSS signals (similar to tuning hyperparameters on a validation set in supervised learning). The filter reduces the average localization error to below $\sim$0.1 meters.

1.5. PID Controller

In [7] the PID controller converts the waypoints into steering by computing the car's angle towards the average between the first two waypoints. The angle is then input to a PID controller to minimize. Computing the angle based on entangled waypoints makes the steering angle towards the path depend on the predicted speed of the vehicle. This works well at the low driving speeds of [7] but becomes hard to tune when driving at higher (and more diverse) speeds like our expert. Instead, we follow the first waypoint that is at least a certain aim distance a away from the center of the car (or the last one). We keep a similar to [7] and use $a = 2.25$ when driving slower than 5.5 m/s (inside intersections) and $a = 3.0$ when driving faster. Longitudinal control is kept similar, we use the velocity between the waypoint 0.5 second into the future and 1 second into the future as target speed. The models using the disentangled waypoint representation do not require tuning an additional controller. Since we directly predict the input to the expert's PID controller, we can reuse the same PID controllers from the expert (see Section 1.1), which we know work well.

The task of yielding to stop signs is defined in CARLA to stop the vehicle on a STOP paining printed on the road. This stop painting is largely occluded by the motor hut once the car is on the sign because our camera is mounted at the back roof of the vehicle. We observe many cases where TF++ detects the stop sign initially and slows down but starts driving again once the painting becomes occluded. We can address this problem in the controller because TF++ detects stop sign bounding boxes in BEV as auxiliary task. For our final model, we keep the last detected stop sign in a buffer and transform it to the current ego coordinate system using the UKF motion estimation. If the car is on the stop sign bounding box we set the action to brake in the controller until the stop sign is cleared. Table 3 shows the impact

Stop Contr.	DS ↑	RC ↑	Stop ↓
-	60 $\pm$ 5	99 $\pm$ 1	1.35
✓	**70** $\pm$ 6	**99** $\pm$ 0	**0.26**

Table 3: **Stop sign controller.**

of the stop sign controller change on the validation routes. TF++ detects most stop signs and the controller is able to reduce Stop infractions by 5x, leading to an improvement of 10 DS.

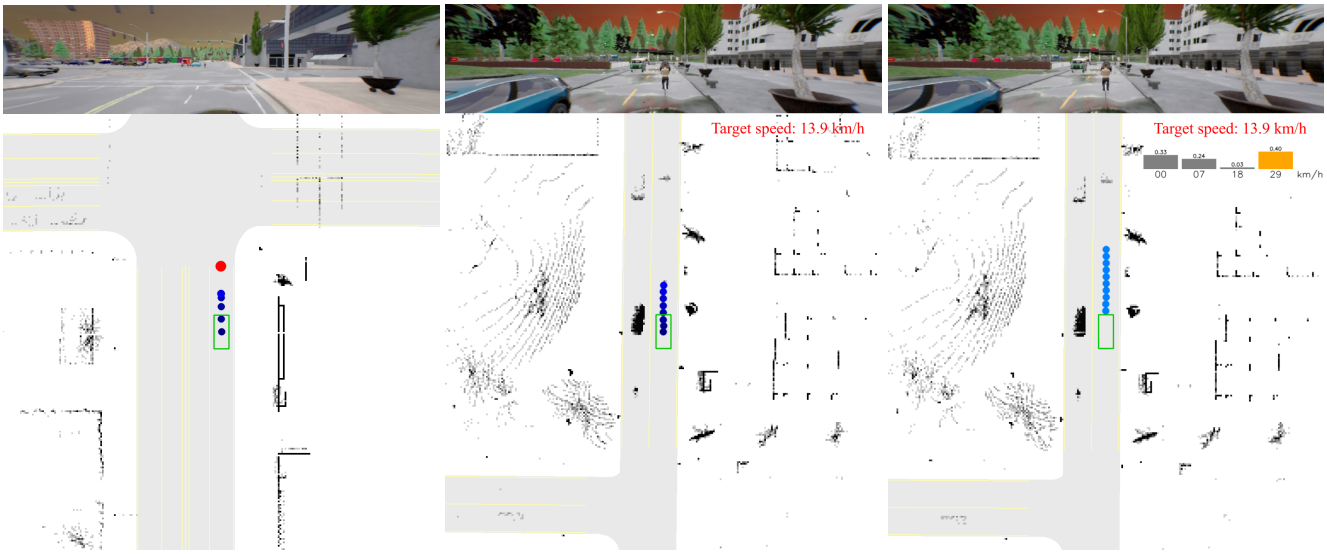
2. TransFuser++ Implementation Details

2.1. Attention Pooling Implementation

The core implementation of our attention pooling with a transformer decoder is similar to [20]. We use the 8x8 features coming from the BEV branch and reduce the number of channels (1512 with our RegNetY-3.2GF backbone) to 256 with a 1x1 convolution. We add a sinusoidal positional embedding to the features and flatten them afterward. The velocity input, we normalize with a 1D Batchnorm and embed it with a 2 layer MLP to 256 features. Afterward, we add a learnable positional encoding. The velocity token then gets concatenated to the BEV tokens. These tokens are then processed by a standard Transformer decoder [21] that has 8 heads, gelu activation function [8], 6 layers and uses layernorm [3]. We use 1 learned queries for every predicted waypoint and 1 additional one if we predict target speeds. The tokens are then fed as inputs into a GRU, whose hidden state is initialized with an (MLP embedded) target point. A linear layer converts the output to 2 dimensions, and a cumulative sum is applied to the final vector (forcing the network to predict offsets from the first point).

2.2. Data Augmentation

When collecting data with the expert driver, we mount an additional camera on the vehicle that collects augmented frames and labels at every time step. This means for every frame in the dataset, we have an augmented counterpart. We shift the camera by ± 1 meter to the left or right (of the vehicle) and rotate it by $\pm 5^\circ$ around the yaw axis. The particular values are drawn from a uniform distribution. During training, we load the perturbed camera with 50% probability. We transform the 2D way point labels and other data such that they have the perturbed camera as center. The shifted and rotated waypoint labels do not describe an actual recovery trajectory that an expert would take, and are instead the original waypoints in the center of the lane. However, the PID controller will steer the car back to the center of the lane, when the network predicts these augmented waypoints, achieving a similar effect. The range of errors the



(a) Slowing down at a green light. (b) Slowing down behind a cyclist. (c) Disentangling route and target speed.

Figure 3: **Waypoints are ambiguous.** The model's output representation forces it to predict a single mode for future velocities.

network is trained to recover from equals the range of the perturbations. To generate the perturbed camera, we need access to a rendering engine (CARLA in our case). On real data, a similar effect can be achieved using novel view synthesis [2].

3. Additional Results

3.1. Longest6 Ablations

As a sanity check, we test our additions to TransFuser by repeating the experiments on the training towns (Longest6) while training with all data. The results are presented in Table 5. We start with our reproduced version of TransFuser and iteratively add 1 change in every row. All results are the average of 3 training seeds, each evaluated 3 times. The reported standard deviation is across the training seeds. The main problems of the reproduced TransFuser are high collisions with other vehicles (Veh) and collisions with the static environments (Stat). Replacing the global average pooling and MLP with a Transformer decoder improves the driving score by 17 points, improving both vehicle collisions and collisions with the environment (indicating improved steering). Adding shift and rotation augmentations has a similar effect, improving the driving score by +9 and reducing collisions. The disentangled representation has a similar effect than on the validation routes, increasing vehicle collisions and reducing collisions with the environment. Since Longest6 has denser traffic, the increased vehicle collisions have a larger impact here, leading to an overall decrease in 2 DS. Two stage training improves DS by +3 and scaling the dataset by 3x leads to a result of 69 DS. Since the waypoint representation is slightly better on longest6, we additionally report the result of a variant called TF++ WP in Table 5. It was trained with the released dataset and uses waypoints as output representation, but is otherwise identical to TF++. TF++ WP again has a slightly higher DS of +1. We also report the result of an ensemble of the 3 training seeds following prior work [7]. The ensemble reduces vehicle collisions by 0.10 and leads to a SotA result of 73 DS on longest6.

3.2. Brake Threshold

The confidence weighted PID controller from the models using target speed prediction has a hyperparameter that determines at which confidence threshold the action is set to full brake. In Table 6 we investigate the effects of different thresholds on validation towns.

Lower thresholds reduce vehicle collisions at the cost of lower route completion due to false positive braking. Overall, the choice of threshold is robust, only changing by 3 DS between the best and worst tested threshold. We use the default threshold of 50% for all validation town experiments. On Longest6, the brake threshold has a slightly larger impact because the dense traffic puts a higher focus on collision avoidance. We therefore use the threshold 33% for Longest6 experiments.

3.3. Additional Baselines

We report 3 additional results on the validation towns in Table 4. TF++ (all towns) is TF++, but its training includes the validation towns. This is not a fair comparison to other methods, and rather serves to illustrate what part of the re-

Method	DS ↑	RC ↑	IS ↑	Ped ↓	Veh ↓	Stat ↓	Red ↓	Stop ↓	Dev ↓	TO ↓	Block ↓
LAV v2 [5]	27 ± 1	98 ± 1	0.27 ± 0.01	**0.00**	**0.35**	0.04	2.89	1.42	**0.00**	0.09	**0.00**
Perception PlanT [18]	37 ± 5	86 ± 7	0.45 ± 0.09	**0.00**	0.92	0.31	0.09	1.87	**0.00**	0.25	0.14
TransFuser (ours)	39 ± 9	84 ± 7	0.46 ± 0.06	**0.00**	0.74	1.04	0.20	1.07	**0.00**	0.23	0.21
TCP [23]	58 ± 5	85 ± 3	0.67 ± 0.06	**0.00**	**0.35**	0.16	**0.01**	1.05	**0.00**	0.19	0.19
TF++ (ours)	**70** ± 6	**99** ± 0	**0.70** ± 0.06	0.01	0.63	**0.01**	0.04	**0.26**	**0.00**	**0.05**	**0.00**
TF++ (all towns)	90 ± 1	99 ± 1	0.91 ± 0.02	0.00	0.18	0.00	0.05	0.00	0.00	0.02	0.00
Expert	*94*	*95*	*0.99*	*0.00*	*0.02*	*0.00*	*0.02*	*0.00*	*0.00*	*0.00*	*0.08*

Table 4: **Performance on validation towns (LAV).** Reproduced models, std over 3 trainings and 3 evaluations.

Method	DS ↑	RC ↑	Veh ↓	Stat ↓
TransFuser (ours)	40 ± 3	82 ± 2	1.17	0.57
+ transformer decoder	57 ± 3	90 ± 3	0.93	0.19
+ data augmentation	66 ± 4	94 ± 2	0.64	0.07
+ disentangled	64 ± 4	96 ± 1	0.88	0.02
+ two stage	67 ± 2	96 ± 1	0.82	0.01
+ 3x data	69 ± 1	**97** ± 1	0.79	**0.00**
TF++ WP (ours)	70 ± 4	94 ± 2	0.66	0.01
+ ensemble	**73**	**97**	**0.56**	0.01

Table 5: **Longest6 Ablations.** Each row has the previous one as baseline. Std over 3 training and 3 evaluation runs.

Threshold	DS ↑	RC ↑	IS ↑	Veh ↓
50%	60 ± 6	**98** ± 1	0.61 ± 0.06	0.73
40%	58 ± 1	96 ± 5	0.61 ± 0.04	0.78
33%	**61** ± 3	96 ± 4	**0.64** ± 0.07	**0.61**
25%	59 ± 3	95 ± 6	0.63 ± 0.08	0.64

Table 6: **Brake threshold.**

maining problems are due to generalization issues. Including the validation towns during training increases the driving score by 20 points, improving both vehicle collisions and stop sign infractions. TF++ is close to expert level performance in this setting, "underfitting" by 4 DS. The expert still has slightly lower vehicle collisions (Veh).

We retrain Perception PlanT [18] on its released dataset. It achieves a DS of 37. Perception Plant uses a handcrafted intermediate representation (bounding boxes) as visual abstraction. This makes the method more interpretable, but the downside of human designed representations are that they might miss important things. In this case, the bounding boxes do not include stop signs, leading to the method ignoring all stop signs (Stop) and incurring a large penalty for that infraction (1.87). Stop signs were not considered in the benchmark this method was developed on (Longest6).

The reproduced LAV v2 [5] also achieves a surprisingly low driving score of 27. The reason for this are its high stop sign and red light infractions. The red light infractions occur almost exclusively in Town 02 which has European style traffic lights. We evaluate on the LAV routes with additional scenarios (7,8,9,10) compared to the original benchmark. Due to the added scenarios, many traffic lights will turn yellow (and then red) just as the agent approaches an intersection. LAV v2 ignores those situations, incurring many red light infractions, bringing the overall score down. Besides the failure to adhere to traffic rules, LAV v2 achieves SotA results in route completion and vehicle collision avoidance.

3.4. Additional Examples

Fig. 1 shows another example of the importance of the target point for recovery. This time we forcefully steer the ego car onto the sidewalk, which is also an out-of-distribution situation. The target point conditioned methods (here TransFuser and TCP) extrapolate their waypoints towards the nearby target point and drive back to the center of the lane. The discrete conditioned TransFuser that does not have access to the geometric information of the target point gets stuck on the sidewalk instead.

Fig. 2 shows another example of harmful target point extrapolation. TCP and TransFuser both predict waypoints leaving the road in a right turn where the target point is far behind the turn. Changing the global average pooling + MLP approach in TransFuser to a transformer decoder mitigates the problem.

3.5. Additional Experiments

Multimodal waypoints: Instead of disentangling the velocity from the waypoints one could also allow the network to predict multiple sets of waypoints as is sometimes done in trajectory forecasting. To test this approach we train a model with two waypoint GRUs and a selection head to classify the better mode. During training we compute the loss as the minimum L1-loss from both predicted waypoints. The classification head is supervised with a binary cross entropy loss to classify which of the two waypoint losses has the lower L1 loss. Table 7 shows that the representation performed 2 DS worse than using standard unimodal waypoints. This is not a big difference but the multi-

Output	DS ↑	RC ↑	Veh ↓	Stat ↓
Multimodal WP	47 ± 8	**96** ± 1	1.13	0.20
Unimodal WP	**49** ± 8	90 ± 4	**0.70**	**0.10**

Table 7: **Multimodal waypoint representation.**

modal waypoints are more complex so there is not really a reason to use them.

NC conditioned AIM: To test whether recovery from the target point bias depends on the architecture, we reproduce the AIM [15] approach. Compared to TransFuser it uses no LiDAR, no auxiliary losses and no transformers, but has the same target point conditioned waypoint GRU as decoder. Like with TransFuser we run two variants, one with TP conditioning and one with NC. The result presented in Table 8

Cond.	DS ↑	RC ↑	Dev ↓
NC	**27** ± 2	50 ± 3	0.96
TP	26 ± 2	**86** ± 8	**0.00**

Table 8: **Conditioning effect on AIM [15]**

also show a strong impact of TP conditioning on RC and route deviations (Dev) for the AIM architecture, suggesting that the target point bias does not depend on the particular architecture used. Consistent with [15] we also observe that AIM performs worse than TransFuser overall (-13 DS).

Target Point statistics: To show the importance of the TP for the network predictions we train a model where we input the TP as a TF decoder token so that we can analyse the attention weights. We average the attention across all layers, heads and waypoint queries. We run the model on the validation routes and observe an avg. of 25% attention on the TP token which is 16.5x higher than uniform. We also test how steering correlates with the target point. The sign of the steering angle (when larger than 1°), of our final model, is the same as that of the target point (TP) 92% of the time (93% for the expert) on the validation routes.

3.6. CARLA Leaderboard

The CARLA leaderboard [1] (we are considering version 1 in this discussion) is a test server where groups can submit agents enclosed within a docker container. These agents are then evaluated on a set of 10 routes across 2 secret towns. Each route is traversed 2× with different weather conditions. Additionally, these 20 routes are then repeated 5 times with different random seeds and average metrics across all routes are reported to the user. The CARLA leaderboard serves as a standardized evaluation platform, comparable to a test set, and some works solely rely on it to compare to other work [5, 20, 23]. Significant progress has been achieved on this benchmark. Driving scores have increased from approximately 10 DS to 70-80 DS over a span of three years. The top performing methods on the leaderboard have released code and models alongside the release of their research papers. In this study, we try to reproduce the top 4 reported results by submitting the released code to the leaderboard, either with the released model (*) file or a retrained one (†).

Method	Reproduced			Reported		
	DS ↑	RC ↑	IS ↑	DS ↑	RC ↑	IS ↑
LAV* [5]	~~25~~	~~46~~	~~0.74~~	62	**94**	0.64
Interfuser* [20]	34	75	0.45	**76**	88	0.84
TCP † [23]	48	66	0.77	70	83	**0.85**
TF++ (ours)	53	71	0.76	-	-	-
TransFuser † [7]	55	**90**	0.63	61	87	0.71
TF++ WP (ours)	62	78	0.81	-	-	-
TF++ WP Ens. (ours)	**66**	79	**0.84**	-	-	-

Table 9: **CARLA leaderboard reproduced results.**

Table 9 shows significant disparities between reported results and the outcomes obtained when utilizing the official codebases. Notably, the released model of the SotA method Interfuser achieves less than half of the reported score. With the exception of TransFuser, these differences are larger than what we would expect from training or evaluation variance. The CARLA leaderboard ensures repeatability (within the bounds of evaluation variance) because one can rerun the submitted docker container. It does however not guarantee reproducibility, since released code and models can differ from what was used to achieve the reported score. Currently, it is publicly not known how to achieve scores above 60 DS, even though some groups have achieved such scores in the past. Given that the official codebases incorporate the main concepts discussed in the respective papers, these results indicate that factors not explicitly emphasized in the papers significantly influence the outcomes observed on the CARLA leaderboard.

Note, that it has been documented in [7] (Table 5) that some changes can have a significant impact on performance on the leaderboard (+19 DS), even though the same changes do not generalize to the public towns (-2 DS on Longest6). This discrepancy can be attributed, in part, to substantial fluctuations in RC, which we don't observe in the publicly available towns. The evaluation is secret, so the exact reasons for these fluctuations are unclear. The LAV results are crossed out because the low score can be attributed to CUDA out of memory errors stemming from the released software, that appear after route 53. The CARLA leaderboard fails silently and gives 0 DS on routes where the software crashes. We contacted the organizers in this case because some of the auxiliary metrics looked unusual.

Furthermore, the auxiliary metrics on the CARLA leaderboard are typically not reported because they are

known to be incorrectly computed[1]. The results presented in Table 9 were obtained with released code and models around April 2023. It should be noted that these results are subject to change if the authors decide to update their repositories. We submit TF++ and TF++ WP (see Section 3.1) to the CARLA leaderboard. Surprisingly, we observe a large difference of 9 DS between the representations. The reason for that difference is unclear because the effect is not reproducible on the publicly available towns. Similar to prior work [7], we submit an ensemble with 3 training seeds of our best model to the leaderboard. The ensemble increases the driving score by 4 points and is the best publicly available model, at the time of writing.

Discussion: The CARLA leaderboard aimed at reproducing success in past benchmarks like ImageNet for autonomous driving. It has led to similar fast progress, although the resulting methods are not always reproducible and not very well understood. There are structural changes the community could make that we think should be considered when designing future benchmarks.

The CARLA leaderboard does not have an official validation benchmark. Since validation is fundamental to machine learning development, authors have proposed various validation benchmarks [5–7,15,25] using the publicly available towns. They differ along various axes: the towns, route length, scenarios and wheathers used for validation. Based on our experience, none of the available validation benchmarks can reliably predict performance on the CARLA leaderboard. It is hard for authors to make validation routes that are well aligned because the test routes are secret. As a consequence, results on the CARLA leaderboard are often unexpected, requiring authors to run additional investigations [7]. Other authors choose not to use the leaderboard in their publications [9, 12, 18, 24, 25]. Submissions on the CARLA leaderboard may encounter a pending status if all servers are in use. During this project, we have encountered pending times of up to 3 weeks. Submissions themselves then take more than four days to evaluate, leading to total evaluation times of up to 4 weeks. For comparison, we can run a similar amount of simulation on our cluster in 3–6 hours, by evaluating all routes in parallel. Achieving this efficiency requires a scalable infrastructure but uses the same amount of computational resources as when evaluating routes sequentially. We encourage organizers of future testing benchmarks to release an aligned public validation benchmark that authors can use to do rapid experiments on.

The CARLA leaderboard is a benchmark that is not associated with any dataset. Consequently, authors are required to create their own datasets for training their methods. In the early stages, the quality of these datasets, including sensor configurations, label accuracy, and scale, was often subpar. Substantial progress has been made by enhancing the quality of the data itself [11]. Most papers, including this one, introduce a new dataset alongside their method. While this practice allows for innovation in dataset curation techniques, it also complicates fair comparisons between different methods. Additionally, it can obscure the source of performance improvements if the dataset is not studied as well. Although there have been some efforts in dataset curation [4,25], such efforts could be more fruitful if there was a standardized dataset to compare to.

[1] https://github.com/carla-simulator/leaderboard/issues/117

References

[1] Carla autonomous driving leaderboard. https://leaderboard.carla.org/, 2020. 7

[2] Alexander Amini, Tsun-Hsuan Wang, Igor Gilitschenski, Wilko Schwarting, Zhijian Liu, Song Han, Sertac Karaman, and Daniela Rus. Vista 2.0: An open, data-driven simulator for multimodal sensing and policy learning for autonomous vehicles. In *Proc. IEEE International Conf. on Robotics and Automation (ICRA)*, 2022. 5

[3] Lei Jimmy Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *arXiv.org*, 1607.06450, 2016. 4

[4] Dian Chen, Vladlen Koltun, and Philipp Krähenbühl. Learning to drive from a world on rails. In *Proc. of the IEEE International Conf. on Computer Vision (ICCV)*, 2021. 1, 4, 8

[5] Dian Chen and Philipp Krähenbühl. Learning from all vehicles. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2022. 3, 6, 7, 8

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

[7] Kashyap Chitta, Aditya Prakash, Bernhard Jaeger, Zehao Yu, Katrin Renz, and Andreas Geiger. Transfuser: Imitation with transformer-based sensor fusion for autonomous driving. *IEEE Trans. on Pattern Analysis and Machine Intelligence (PAMI)*, 2022. 1, 2, 3, 4, 5, 7, 8

[8] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv.org*, 1606.08415, 2016. 4

[9] Anthony Hu, Gianluca Corrado, Nicolas Griffiths, Zak Murez, Corina Gurau, Hudson Yeo, Alex Kendall, Roberto Cipolla, and Jamie Shotton. Model-based imitation learning for urban driving. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022. 8

[10] Martin Isenburg. Laszip: lossless compression of lidar data. *Photogrammetric engineering and remote sensing*, 2013. 2

[11] Bernhard Jaeger. Expert drivers for autonomous driving. Master's thesis, University of Tübingen, 2021. 8

[12] Xiaosong Jia, Penghao Wu, Li Chen, Jiangwei Xie, Conghui He, Junchi Yan, and Hongyang Li. Think twice before driving: Towards scalable decoders for end-to-end autonomous driving. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2023. 8

[13] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *Proc. of the International Conf. on Learning Representations (ICLR)*, 2019. 2

[14] Rudolph Van Der Merwe. *Sigma-point Kalman filters for probabilistic inference in dynamic state-space models*. PhD thesis, Oregon Health & Science University, 2004. 4

[15] Aditya Prakash, Kashyap Chitta, and Andreas Geiger. Multi-modal fusion transformer for end-to-end autonomous driving. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2021. 7, 8

[16] Ilija Radosavovic, Raj Prateek Kosaraju, Ross B. Girshick, Kaiming He, and Piotr Dollár. Designing network design spaces. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2020. 3

[17] Sashank J. Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of adam and beyond. In *Proc. of the International Conf. on Learning Representations (ICLR)*, 2018. 2

[18] Katrin Renz, Kashyap Chitta, Otniel-Bogdan Mercea, Almut Sophia Koepke, Zeynep Akata, and Andreas Geiger. Plant: Explainable planning transformers via object-level representations. In *Proc. Conf. on Robot Learning (CoRL)*, 2022. 6, 8

[19] Labbe Roger. Kalman and bayesian filters in python. 2020. 4

[20] Hao Shao, Letian Wang, RuoBing Chen, Hongsheng Li, and Yu Liu. Safety-enhanced autonomous driving using interpretable sensor fusion transformer. In *Proc. Conf. on Robot Learning (CoRL)*, 2022. 4, 7

[21] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NIPS)*, pages 5998–6008, 2017. 4

[22] Eric A Wan and Rudolph Van Der Merwe. The unscented kalman filter for nonlinear estimation. In *Proc. IEEE Adaptive Systems for Signal Processing, Communications, and Control Symposium*, 2000. 4

[23] Peng Wu, Xiaosong Jia, Li Chen, Junchi Yan, Hongyang Li, and Yu Qiao. Trajectory-guided control prediction for end-to-end autonomous driving: A simple yet strong baseline. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022. 2, 3, 6, 7

[24] Jimuyang Zhang, Zanming Huang, and Eshed Ohn-Bar. Coaching a teachable student. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2023. 8

[25] Zhejun Zhang, Alexander Liniger, Dengxin Dai, Fisher Yu, and Luc Van Gool. End-to-end urban driving by imitating a reinforcement learning coach. In *Proc. of the IEEE International Conf. on Computer Vision (ICCV)*, 2021. 1, 3, 8