Supplementary Material for
NEAT: Neural Attention Fields for End-to-End Autonomous Driving

Kashyap Chitta*[1,2] Aditya Prakash*[1] Andreas Geiger[1,2]
[1]Max Planck Institute for Intelligent Systems, Tübingen [2]University of Tübingen
{firstname.lastname}@tue.mpg.de

Abstract

*In this **supplementary document**, we begin by providing additional information regarding our architecture (Section 1). We go on to describe our data generation pipeline (Section 2), compare the baselines used in our experiments to their implementation in the original publications (Section 3), and provide details regarding the hyper-parameter choices for all the models (Section 4). Finally, we present additional results and analysis to supplement our findings in the main paper (Section 5). The **supplementary video** contains qualitative driving visualizations of NEAT.*

1. Architecture Details

In this section, we first provide the pseudo-code for the forward pass of our NEAT model for a given query point (Algorithm 1), to supplement the description from the main paper. This is followed by in-depth descriptions of our architecture components, in particular (1) the transformer block of the encoder, (2) the conditional batch normalization operation used in NEAT and the decoder, and (3) the low-level PID controller.

1.1. Transformer

Transformers [23] were developed for natural language processing (NLP) as an alternative to sequence-to-sequence models and subsequently applied to visual domains [3, 10, 20, 22]. A transformer takes as input a sequence of discrete tokens, each represented by a feature vector. The feature vector is supplemented by a positional embedding to incorporate positional inductive biases. These positional embeddings can be fixed, e.g. order of the input tokens in a text or video, or can be learned, so that the network infers any spatial dependencies between tokens during training.

Formally, we denote the input sequence as $\mathbf{F}^{in} \in \mathbb{R}^{N \times D_f}$, where N is the number of tokens in the sequence and each token is represented by a feature vector of dimensionality D_f. The transformer uses linear projections for computing a set of queries, keys and values ($\mathbf{Q}$, $\mathbf{K}$ and $\mathbf{V}$),

$$\mathbf{Q} = \mathbf{F}^{in}\mathbf{M}^{q},\ \mathbf{K} = \mathbf{F}^{in}\mathbf{M}^{k},\ \mathbf{V} = \mathbf{F}^{in}\mathbf{M}^{v} \tag{1}$$

where $\mathbf{M}^{q} \in \mathbb{R}^{D_f \times D_q}$, $\mathbf{M}^{k} \in \mathbb{R}^{D_f \times D_k}$ and $\mathbf{M}^{v} \in \mathbb{R}^{D_f \times D_v}$ are weight matrices. It uses the scaled dot products between $\mathbf{Q}$ and $\mathbf{K}$ to compute the attention weights and then aggregates the values for each query,

$$\mathbf{A} = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^{T}}{\sqrt{D_k}}\right)\mathbf{V} \tag{2}$$

where the softmax is used to normalize the weights. The scaling factor $\frac{1}{\sqrt{D_k}}$ is used to prevent numerical instabilities. Finally, the transformer uses a set of non-linear transformations to calculate the output features, $\mathbf{F}^{out}$ which are of the same shape as the input features, $\mathbf{F}^{in}$.

$$\mathbf{F}^{out} = \text{MLP}(\mathbf{A}) + \mathbf{F}^{in} \tag{3}$$

*indicates equal contribution

Algorithm 1: Pseudo-code for processing a query point with NEAT.

input : $\mathcal{X}, v, \mathbf{p} = (x, y, t, x', y')$; // sensor data, velocity and query point
output: $\{\mathbf{s}_i, \mathbf{o}_i\}_{i=1}^{N}$; // semantic class and (x, y) offset to waypoint at $\mathbf{p}$

$\mathbf{c} = e_\theta(\mathcal{X})$; // encoder
$\mathbf{c}_0 = \text{mean}(\mathbf{c})$; // initial attention
for $i \leftarrow 1$ **to** N **do**
  $\mathbf{a}_i = a_\phi(\mathbf{p}, \mathbf{c}_{i-1})$; // neural attention field
  $\mathbf{c}_i = \text{softmax}(\mathbf{a}_i)^\top \cdot \mathbf{c}$; // update
  $\{\mathbf{s}_i, \mathbf{o}_i\} = d_\psi(\mathbf{p}, \mathbf{c}_i)$; // decoder

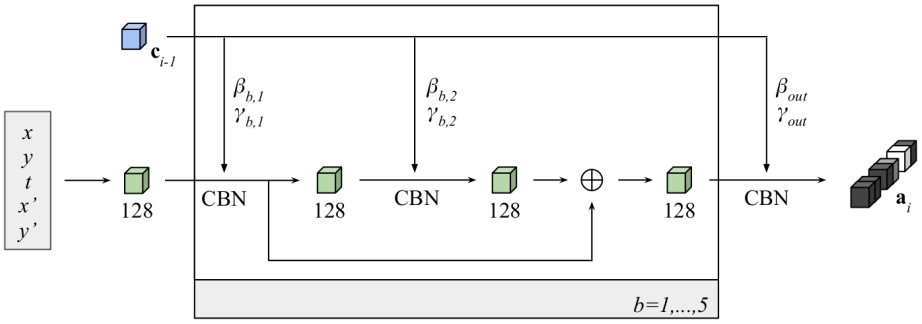

Figure 1: **NEAT implicit function architecture.** NEAT outputs an attention map $\mathbf{a}_i$. The input to NEAT is a query point $\mathbf{p} = (x, y, t, x', y')$, which is processed through an MLP with 5 fully-connected ResNet blocks, and conditioned on $\mathbf{c}_{i-1}$ through conditional batch normalization (CBN, Eq. (4)). The decoder uses an identical network structure as NEAT, but takes as input $\mathbf{c}_i$ instead of $\mathbf{c}_{i-1}$, and uses two output CBN layers which output semantics $\mathbf{s}_i$ and offsets $\mathbf{o}_i$.

The transformer applies the attention mechanism multiple times throughout the architecture resulting in L attention layers. Each layer in a standard transformer has multiple parallel attention 'heads', which involve generating several $\mathbf{Q}$, $\mathbf{K}$ and $\mathbf{V}$ values per $\mathbf{F}^{in}$ for Eq. (1) and concatenating the resulting values of $\mathbf{A}$ from Eq. (2).

1.2. Conditional Batch Normalization

We employ the same MLP architecture for the NEAT and decoder in all experiments, illustrated in Fig. 1. The query points $\mathbf{p} = (x, y, t, x', y')$ are passed through a fully-connected layer to obtain a 128-dimensional feature per point. This is then passed through 5 ResNet [13] blocks ($b = 1, ...5$). Each block involves a composite operation of (1) a conditional batch normalization (CBN) function [8, 11], (2) a ReLU activation, and (3) a linear layer. This composite operation is applied two times followed by an additive skip connection. The output is passed through a final CBN layer and ReLU activation before being projected to the required output dimensions ($\mathbf{a}_i \in \mathbb{R}^{S*T*P}$ for NEAT, and two outputs $\mathbf{s}_i \in \mathbb{R}^K$ and $\mathbf{o}_i \in \mathbb{R}^2$ for the decoder from two different output CBN layers). The attention map $\mathbf{a}_i$ and semantic outputs $\mathbf{s}_i$ use the softmax activation, whereas no extra activation function is required for the offset outputs $\mathbf{o}_i$.

To implement CBN, we first pass the feature ($\mathbf{c}_{i-1}$ in Fig. 1) through a 2-layer ReLU MLP to obtain 128-dimensional vectors $\beta_{b,1}$, $\beta_{b,2}$, $\gamma_{b,1}$, and $\gamma_{b,2}$ for each CBN operation in the network. Given the input feature f_{in}, CBN is performed as a standard batch normalization operation with these values of β and γ obtained from $\mathbf{c}_{i-1}$:

$$f_{out} = \gamma \frac{f_{in} - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta \tag{4}$$

where μ and σ are the empirical mean and standard deviation of the input features f_{in} over the batch dimension. At test time, a running mean of μ and σ computed during training is used to fix these values. ϵ is used to prevent numerical instabilities, set to 10^{-5} (the default value in PyTorch [17]).

Algorithm 2: Generating an action from waypoints.

input : $v, (x', y'), \mathbf{r}, \{\mathbf{w}_t\}_{t=T+1}^{T+Z}$; `// velocity, red light indicator, target and waypoints`
output: steer $\in (-1, 1)$, throttle $\in (0, 1)$, brake $\in (0, 1)$; `// vehicle control`

$\delta = 0$
for $t \leftarrow T + 1$ **to** $T + Z - 1$ **do**
  δ += $\frac{1}{Z-1}||\mathbf{w}_{t+1} - \mathbf{w}_t||$; `// desired velocity`
$\alpha^* = \tan^{-1}\left(\frac{(\mathbf{w}^*)[1]}{(\mathbf{w}^*)[0]}\right)$; `// aim`
$\alpha_b = \tan^{-1}\left(\frac{y'}{x'}\right)$; `// backup aim`
if $||\alpha^* - \alpha_b|| > \beta_{err}$ **or** $\alpha_b < \alpha^*$ **then**
  $\alpha = \alpha_b$
else
  $\alpha = \alpha^*$
steer = LatPID(α)
if $\mathbf{r} = 1$ **or** $\delta < \beta_{min}$ **or** $\delta < v\beta_{ratio}$ **then**
  throttle $= 0$
  brake $= 1$
else
  throttle = LonPID($\delta - v$)
  brake $= 0$

1.3. PID Controller

To convert waypoints to vehicle controls, we use a combination of two PID controllers. This process is summarized in Algorithm 2. We first compute the $Z-1$ vectors between waypoints of consecutive time-steps. The mean magnitude of these vectors is input to longitudinal controller (LonPID in Algorithm 2), which tries to match the vehicle velocity v to the desired velocity δ as much as possible. We then identify the waypoint closest to 4 meters in distance from the ego-vehicle as $\mathbf{w}^*$. This is used to compute the angular aim, which is input to a lateral controller (LatPID). Each controller has 3 gain parameters, which are tuned on a subset of our training routes. For the speed controller, we set $K_p = 5.0, K_i = 0.5, K_d = 1.0$ and for the turn controller, we set $K_p = 0.75, K_i = 0.75, K_d = 0.3$. Both controllers use a buffer of size 40 to approximate the integral term as a running average. The other parameters used are a brake threshold speed β_{min}, angular error threshold β_{err}, and brake speed ratio β_{ratio}, which we tune jointly with the gain parameters, and set to 0.4, 0.2, and 1.1 respectively.

2. Data Generation and Evaluation

In this section, we describe our expert and data generation pipeline; providing statistics regarding the distribution of navigational commands in our dataset. We then describe the routes and scenarios used in our novel evaluation setting. Note that our data generation process closely mirrors that of [19], with the key difference being the use of additional weather and daylight conditions during training.

2.1. Sensor Configuration

We use 3 cameras, each with a field of view (FOV) of 100°, mounted at a height of 2.3m from the ground level and 1.3m in front of the centroid of the ego-vehicle. We offset the cameras forward by 1.3m to avoid the hood of the ego-vehicle from occluding a portion of the rendered images. The cameras are oriented forward, 60° left and 60° right respectively. We extract images at a resolution of 400 × 300 pixels. When a 256 × 256-pixel crop is taken from these 3 images, they provide a full 180° field-of-view of the scene, which is useful for the BEV semantic prediction task. For the ground-truth depth and semantic maps, we use CARLA's 2D depth and semantic segmentation cameras[1] placed at the same location as the three RGB cameras. Additionally, we extract the BEV semantics for NEAT using a semantic segmentation camera with a resolution of 512 × 512 pixels, placed at a height of 100m from the center of the ego-vehicle with a FOV of 50°. We use

[1] https://carla.readthedocs.io/en/latest/ref_sensors

an IMU to obtain orientation, a GPS sensor to register frames to the current time-step during training, and a speedometer to get the current velocity of the ego-vehicle which is used as an input to our encoder.

2.2. Expert

For generating training data, we roll out an expert policy designed to drive using privileged information from the simulator. We adapt the code[2] provided by the authors of [2] to build our expert. Several heuristics based on the global position of the dynamic agents and traffic lights are used by this expert to avoid collisions and traffic violations. We make two modifications to the existing code, (1) we add stop sign avoidance functionality, and (2) we change the distance and speed heuristics used for avoiding collisions and following traffic lights to make the expert more cautious. Both changes improved the driving scores of the expert in our preliminary analysis.

Given the route to be followed as a sequence of sparse waypoints, the expert uses an A* planner to interpolate these sparse waypoints to dense waypoints in the CARLA town map with a sampling resolution of 1m. From these dense waypoints, it selects two waypoints - the nearest waypoint after 4m from its current position as the near node and the nearest waypoint after 7m from its current position as the far node. It uses the orientation of the near node as the aim direction, which is fed to the lateral PID controller, with the steering value as its output. Note how we designed the PID controller for NEAT to mimic this at test time, by choosing the waypoint closest to 4m to determine the aim for steering. The expert uses the orientation of the far node as an indication of an approaching turn. For longitudinal control, the expert uses a discrete set of speed values, $\{0, 4, 7\}$m/s as the target speed depending on its current circumstances. If the expert is close to pedestrians, vehicles, red light, or an uncleared stop sign (which is accessed as privileged information from CARLA), then the target speed is set to 0 m/s. If the expert has to steer by more than 5° or is approaching a turn, as indicated by the orientation of the far node, then the target speed is set to 4 m/s. Under every other circumstance, the target speed is set to 7 m/s. A longitudinal PID controller then tries to match the speed of the expert policy to this target speed as much as possible. For the longitudinal controller, we set $K_p = 5.0, K_i = 0.5, K_d = 1.0$ and for the lateral controller, we set $K_p = 1.25, K_i = 0.75, K_d = 0.3$, following the default implementation from [2]. Both controllers use a buffer of size 40 for the integral terms.

2.3. Training Routes

The CARLA simulator repository contains a recommended set of 50 routes for training and 26 routes for testing, released accompanying a previous version of the simulator for the 2019 CARLA challenge[3]. Each route is defined by a sequence of waypoints, from a total of 6 towns in CARLA. However, these routes are heavily biased towards driving straight (Fig. 2) due to which the models trained with data generated on these routes are likely to have lower performance at turns and intersections. To overcome this limitation, we generate a set of additional routes which incorporate more turns and intersections.

Towards this goal, we first use the positions of the traffic lights to sample the intersections present in the town map, since most of the intersections contain traffic lights. For each such sampled intersection, we generate multiple pairs of start and end points, by sampling points at a distance of 100m from the intersection along the canonical axes (road segments at intersections in CARLA are aligned with the canonical axes in world coordinates). From this set of sampled points, we select pairs of points that lie on different road segments of the intersections. Using this mechanism, we sample 8 pairs of turn routes at each intersection. However, if any of these sampled points do not lie on a road segment, e.g. in 3-way intersections, the simulator automatically maps the point lying off the road segment to the nearest point on a road segment. This results in several long routes encompassing multiple intersections.

While this method improves the relative representation of turns by 2-3% in the dataset, the turns are still severely under-represented in comparison to the 'Follow Lane' command. Therefore, we sample additional turns at these intersections. For this, we interpolate each of the routes (using the same procedure followed by the expert, as explained in Section 2.2) obtaining a dense set of waypoints along each. Then, we explicitly sample points along these routes where the navigational command provided by CARLA changes from 'Follow Lane' to 'Turn Left', 'Turn Right' or 'Go Straight'. These points indicate the starting position of an intersection. When the navigation command changes back to 'Follow Lane' along the interpolated route, this indicates the end of an intersection, which we use as the end point. In this manner, we sample several pairs of start and end points along each long route with each pair representing a single intersection of length 25-50m. Adding these short, single-intersection routes significantly improves the distribution of navigational commands in our dataset (Fig. 2).

[2] https://github.com/bradyz/leaderboard/blob/master/team_code/auto_pilot.py
[3] https://github.com/carla-simulator/leaderboard/tree/stable/data

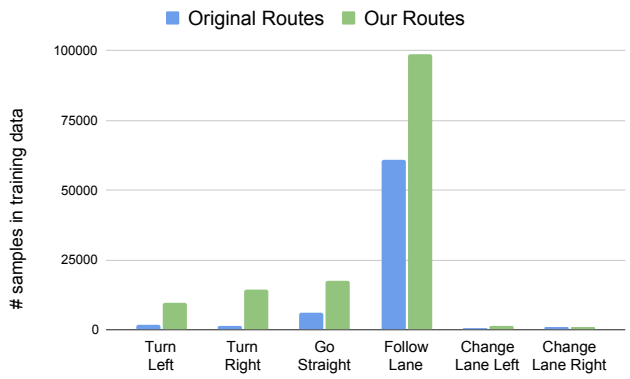

Figure 2: **Distribution of Navigational Commands.** We report the distribution of navigational commands for the original routes provided in the Leaderboard repository and our routes used for generating data. Our routes incorporate more turns and intersections in the training data.

2.4. Training Scenarios

The official CARLA Leaderboard repository also provides a set of scenarios[4] for each town based on the NHTSA pre-crash typology[5]. Each scenario is defined by (1) a trigger 'transform' which indicates the spawn location and the orientation of that scenario in a particular town and (2) 'other actors' which contains information about additional agents present in that scenario. While the CARLA scenario manager provides access to a set of 10 scenarios in total, each assigned a specific scenario number (1-10), the trigger transforms for only 3 of these are publicly released with the Leaderboard starter code - control loss without previous action, obstacle avoidance without prior action and obstacle avoidance with prior action.

To include scenarios such as other vehicles running red lights and unprotected turns, which can be dangerous in a real-world situation, we create a dense set of trigger transforms for additional scenarios while generating training data. We focus on safety-critical scenarios involving pedestrians emerging from occluded regions to cross the road at random locations, vehicles running red lights, and unprotected turns. Specifically, we include Scenarios 1, 3, 4, 7, 8, 9, 10 in our training dataset. Scenario 2 occurs naturally during the driving in the presence of other dynamic agents so we do not explicitly include it. Since we focus on safety-critical scenarios, we also do not include Scenarios 5 and 6, which involve lane changing. To spawn these scenarios densely, we consider the trigger transforms provided in the Leaderboard's set of scenarios and sample additional spawn locations at a 5m distance along the canonical axes in a 10m $\times$ 10m grid centered at each available trigger position. For the spawn orientation, we select yaw $\in \{0°, 90°, 180°, 270°\}$. Therefore, for each provided trigger transform in a particular town, we generate 36 additional trigger transforms for each scenario. Since multiple scenarios can have the same trigger position, we randomly select one scenario per trigger position while generating training data.

2.5. Training Data Statistics

For training data generation, we use all 8 publicly available towns (Town01, Town02, Town03, Town04, Town05, Town06, Town07, Town10) and 7 weather conditions (Clear, Cloudy, Wet, MidRain, WetCloudy, HardRain, SoftRain); storing data at 2 FPS. The number of dynamic agents in each town is then fixed (Town01: 120, Town02: 100, Town03: 120, Town04: 200, Town05: 120, Town06: 150, Town07: 110, Town10: 120). We sample the sun elevation from a mixture of Gaussian distribution with 6 components. The mean values used for these Gaussian components (-80, 0, 5, 15, 35, 75) correspond to 6 different daylight conditions used in our evaluation (Night, Twilight, Dawn, Sunset, Morning, Noon). The sun azimuth is sampled randomly from $\{0, 45, 90, 135, 180, 225, 270, 315\}$ degrees. The weather and daylight conditions are varied every 0.5 seconds in simulation, to obtain a uniform distribution during training. We do not use DAgger [18, 21] during training. Our training set contains 130000 frames (sampled from 'Our Routes' in Fig. 2) and our validation set contains 20000 frames (sampled from the 'Original Routes' in Fig. 2).

[4] https://github.com/carla-simulator/leaderboard/blob/stable/data/all_towns_traffic_scenarios_public.json
[5] https://leaderboard.carla.org/scenarios/

	Town	Weather	Daytime	Length (m)
0	1	HardRain	Twilight	1303
1	1	Clear	Sunset	1303
2	1	Wet	Night	1303
3	1	Clear	Noon	322
4	1	Cloudy	Night	322
5	1	Wet	Morning	322
6	2	Wet	Dawn	110
7	2	WetCloudy	Twilight	110
8	2	WetCloudy	Night	110
9	2	Cloudy	Sunset	638
10	2	MidRain	Sunset	638
11	2	SoftRain	Twilight	638
12	3	MidRain	Noon	306
13	3	Clear	Twilight	306
14	3	HardRain	Dawn	306
15	3	HardRain	Night	462
16	3	WetCloudy	Noon	462
17	3	Cloudy	Dawn	462
18	4	WetCloudy	Dawn	2811
19	4	SoftRain	Morning	2811
20	4	SoftRain	Sunset	2811

	Town	Weather	Daytime	Length (m)
21	4	WetCloudy	Morning	204
22	4	Clear	Morning	204
23	4	MidRain	Night	204
24	6	SoftRain	Noon	686
25	6	SoftRain	Dawn	686
26	6	SoftRain	Night	686
27	6	WetCloudy	Sunset	706
28	6	Wet	Noon	706
29	6	HardRain	Morning	706
30	5	MidRain	Twilight	337
31	5	Clear	Dawn	337
32	5	MidRain	Dawn	337
33	5	Wet	Twilight	1642
34	5	HardRain	Noon	1642
35	5	Clear	Night	1642
36	3	Wet	Sunset	961
37	3	Cloudy	Night	961
38	3	Cloudy	Noon	961
39	4	Cloudy	Morning	225
40	4	MidRain	Morning	225
41	4	HardRain	Sunset	255

Table 1: **Evaluation routes.** Environmental condition and length (in meters) of each of the 42 evaluation routes.

2.6. Evaluation Routes

Our proposed evaluation setting consists of 42 routes from 6 different CARLA towns (Town01-Town06). We consider 14 unique routes: 2 routes from Town01, Town02, Town05, Town06, and 3 routes from Town03, Town04. Each unique route is repeated 3 times, but with a different environmental condition on each repetition, giving the final set of 42 routes. Each route has a unique environmental condition obtained by combining one of 7 weather conditions (Clear, Cloudy, Wet, MidRain, WetCloudy, HardRain, SoftRain) with one of 6 daylight conditions (Night, Twilight, Dawn, Morning, Noon, Sunset). We provide details regarding the weather, daytime, and length of the 42 evaluation routes in Table 1 and visualizations of the 14 unique routes in Fig. 3.

2.7. Evaluation Scenarios

During evaluation, we consider all the 10 scenarios available in CARLA: control loss without previous action, longitudinal control after leading vehicle's brake, obstacle avoidance without prior action, obstacle avoidance with prior action, lane changing to evade slow leading vehicle, vehicle passing dealing with oncoming traffic, crossing traffic running a red light at an intersection, unprotected left turn at an intersection with oncoming traffic, right turn at an intersection with crossing traffic, crossing negotiation at an unsignalized intersection. We use the trigger transforms for these scenarios from the CARLA scenario runner GitHub repository[6]. While the CARLA leaderboard framework[7] considers prioritized sampling of scenarios (scenarios with higher ID are given preference, e.g. scenario 10 is prioritized over scenario 9 and so on) in case multiple scenarios can be spawned at a particular trigger location, we consider uniform sampling in our evaluation.

3. Baselines

In this section, we describe the adapted versions of the baselines CILRS [7] and LBC [2] used in our experiments. In particular, we highlight the updates made to adapt these models from the CARLA version of their original publication (0.8.4 for CILRS and 0.9.6 for LBC) to the latest CARLA version 0.9.10. We then discuss in detail the new baselines designed in this work, AIM-MT (which is based on AIM [19]) and AIM-VA (which is based on Visual Abstractions [1]).

[6] https://github.com/carla-simulator/scenario_runner/blob/master/srunner/data/all_towns_traffic_scenarios.json
[7] https://github.com/carla-simulator/leaderboard

Figure 3: **Visualization of evaluation routes.** We show the 14 unique routes on their corresponding town map. The starting point is shown in red, the destination in blue and the route is highlighted in green.

3.1. CILRS

CILRS follows the conditional imitation learning paradigm [6, 7], where driver intention is encoded and passed into the driving model in the form of a discrete navigational command. While the original architecture used 4 navigational commands (follow lane, left/right/straight at intersection), we adapted this model to include 2 additional commands for lane changes (left and right). Since the space of navigational commands is discrete, CILRS uses a conditional module to select one of several command branches based on the input command. The command branch directly outputs control values (steer, throttle, and brake). The best performing CILRS architecture involves a ResNet34 encoder pre-trained on ImageNet [9], whose output is flattened and combined with the measured vehicle velocity v using fully connected layers. Additionally, CILRS uses ego-velocity prediction as auxiliary supervision. The features from the encoder (before combining with velocity features) in CILRS are used to output a prediction of the current vehicle speed $\hat{v}$, which is compared to the actual vehicle speed with the

velocity loss $\mathcal{L}_{\text{velocity}}$ defined in Eq. 5.

$$\mathcal{L}_{\text{velocity}} = ||v - \hat{v}||_1 \tag{5}$$

3.2. LBC

LBC is a knowledge distillation approach where a teacher model with access to ground truth BEV semantic maps is first trained using expert supervision to predict waypoints, followed by an image-based student model which is trained using supervision from the teacher. The waypoints predicted by the teacher are projected into the image coordinates of the student model to provide supervision. Once the teacher is trained, it can output waypoints for a given input for any arbitrary driver intention (e.g. turning left or right at an intersection). By supervising with this teacher model instead of the original expert, LBC provides ground truth supervision to its student model corresponding to multiple driver intentions at each training iteration. While the original LBC architecture uses only the front camera image as input and also uses a branched conditional architecture similar to CILRS, the authors recently released an updated version of their architecture[8] with 2 major modifications, (1) multi-view camera inputs which are stacked as channels to a ResNet50 encoder (front, left and right images, similar to our NEAT model), and (2) target heatmap as input (instead of the navigational command) which is formed by projecting the target point into front camera image coordinates. We directly use this codebase in our experiments and found that other modifications (eg. using a single camera input or matching the semantic classes considered by the teacher model to those used by NEAT) did not lead to performance improvements.

3.3. AIM-MT

AIM [19] is an improved version of CILRS which uses a GRU decoder to regress waypoints, similar to the architecture of [12]. An important modification is that the GRU takes as an input the BEV target location, similar to NEAT, rather than conditioning with different branches for discrete navigational commands as in CILRS.

The best-performing architecture for AIM involves a ResNet34 encoder, pre-trained on ImageNet, which outputs a 512-dimensional feature vector. This feature vector is passed through an MLP (consisting of 2 hidden layers with 256 and 128 units) to reduce its dimensionality to 64 for computational efficiency before passing it to the auto-regressive waypoint network implemented using GRUs [4]. The update gate of the GRU controls the flow of information encoded in the hidden state to the output and the next time-step. It also takes in the current position and the target location as input, which allows the network to focus on the relevant context in the hidden state for predicting the next waypoint. Following [19], we use a single layer GRU followed by a linear layer which takes in the hidden state and predicts the differential ego-vehicle waypoints $\{\delta\mathbf{w}_t\}_{t=1}^{T}$ for $T = 4$ future time-steps in the ego-vehicle coordinate frame. Therefore, the predicted future waypoints are given by $\{\mathbf{w}_t = \mathbf{w}_{t-1} + \delta\mathbf{w}_t\}_{t=1}^{T}$. The input to the first GRU unit is set to (0,0) since the BEV space is centered at the ego vehicle's position.

We train the network using an L_1 loss between the predicted waypoints and the ground truth waypoints (from the expert), registered to the current coordinate frame (empirically, this leads to better results than an L_2 loss). Our multi-task variants of AIM, which we call AIM-MT, are constructed by adding different auxiliary tasks to AIM during training. Let $\mathbf{w}_t^{gt}$ represent the ground truth waypoint for time-step t, then the loss function for AIM-MT can be described as:

$$\mathcal{L} = \sum_{t=1}^{T} ||\mathbf{w}_t - \mathbf{w}_t^{gt}||_1 + \lambda_b \, \mathcal{L}_{CE}(\mathbf{s}_{bev}, \, \mathbf{s}_{bev}^{gt}) + \lambda_s \, \mathcal{L}_{CE}(\mathbf{s}_{2d}, \, \mathbf{s}_{2d}^{gt}) + \lambda_d \, ||\mathbf{d} - \mathbf{d}^{gt}||_1 \tag{6}$$

Here, $\mathbf{w}_t$ represents the waypoint at time-step t, $\mathbf{s}_{bev}$ represents the BEV semantic map, $\mathbf{s}_{2d}$ represents the 2D semantic segmentation $\mathbf{d}$ represents the 2D depth map, and $\mathcal{L}_{CE}$ is the cross entropy loss. For the base AIM model with no auxiliary supervision, $\lambda_b = \lambda_s = \lambda_d = 0$.

Auxiliary 2D supervision: We use a deconvolutional decoder to predict the 2D semantic segmentation and 2D depth similar to the architecture of [15]. We consider the $8 \times 8 \times 512$ feature map from the penultimate layer of the ResNet34 encoder (before average pooling) and pass it through 6 convolutional and 2 upsampling layers to output the 2D semantic segmentation and 2D depth at a resolution of 256×256. For the semantic segmentation we consider 7 classes: (1) unlabeled (to which we include pixels labeled by CARLA as building, fence, other, pole, vegetation, wall, traffic sign, sky, ground, bridge, rail track, guard rain, traffic light, static, dynamic, water, terrain, yellow light, green light, and stop sign) (2) vehicle, (3) road, (4) red light, (5) pedestrian, (6) lane marking, (7) sidewalk. As we see in Section 5.3, the additional classes used for the

[8] https://github.com/bradyz/2020_CARLA_challenge

2D semantic prediction task (such as lane marking and sidewalk) are crucial to driving performance, with significant drops in scores observed when only using the same 5 classes as NEAT. For the AIM-MT variant with only 2D segmentation, We set $\lambda_b = 0$, $\lambda_s = 1$ and $\lambda_d = 0$ during training. For the best performing AIM-MT variant with both 2D depth and 2D segmentation, we use $\lambda_b = 0$, $\lambda_s = 1$ and $\lambda_d = 10$.

Auxiliary BEV supervision: We use a spatial broadcast decoder [24] to predict the BEV semantic map from the 512-dimensional feature vector. In terms of structure and supervision, this variant of AIM-MT is most comparable to NEAT. Therefore, the semantic classes used by this variant are the same as those used by NEAT: (1) unlabeled, (2) road, (3) obstacle (pedestrian + vehicle), (4) red light, and (5) green light. We set $\lambda_b = 0.1$, $\lambda_s = 0$ and $\lambda_d = 0$ during training for this variant.

3.4. AIM-VA

AIM-VA generates 2D segmentation map representations of the inputs using a 2D semantic segmentation model. These are then fed into an AIM model (without any auxiliary supervision) to regress waypoints. By abstracting away irrelevant details in the RGB image space (such as textures and lighting), using these Visual Abstractions [1] have been shown to improve robustness in CARLA.

We use a ResNet-50 FCN for semantic segmentation, using the same 7 classes as AIM-MT with auxiliary 2D supervision (see Section 3.3). In the AIM component of the AIM-VA model, we use a ResNet-18, which performs the best.

4. Implementation Details

In this section, we discuss the training hyper-parameter choices and schedules. These optimizer-related hyper-parameters were tuned based on the validation loss of each model on a subset of our training dataset. We perform single-GPU training for all models, on GTX 1080Ti GPUs. Our code is implemented in PyTorch [17].

4.1. NEAT

For NEAT training, we use the AdamW optimizer [16], which is a variant of Adam [14]. The ResNet34 in the encoder is pre-trained on ImageNet [9]. The learning rate is set to 10^{-4}, weight decay to 0.01, and Adam beta values to the PyTorch defaults of 0.9 and 0.999. During training, we track the offset loss on a validation partition of the dataset and save the checkpoint with the lowest offset loss after 50 epochs of training with our dataset for online evaluation. We use a batch size of 16 for all variants of NEAT.

4.2. Other Models

We train the LBC model using the codebase provided by the authors, which uses the Adam [14] optimizer with a learning rate of 10^{-4} and weight decay of 0.0. The model is validated after every epoch. In this implementation, the learning rate is reduced by a factor of 2 if the validation loss does not improve for 5 epochs. We train the model for a maximum of 100 epochs with a batch size of 64 and terminate the training if the learning rate goes below 10^{-6} or if the validation loss has not improved for 15 epochs.

The image to segmentation model of AIM-VA is trained on our dataset using the MMSegmentation library[9] using the default hyper-parameters for the ResNet-50 FCN model. For CILRS, AIM, AIM-MT, as well as the segmentation to control model of AIM-VA, we use the same AdamW optimizer hyper-parameters as NEAT. The models are validated after every 5 epochs and we use early stopping to terminate the training if the validation loss has not improved for 25 epochs, or at a maximum of 100 epochs. For CILRS, the weight for the velocity loss term is set to 0.05 and batch size is set to 256. We use a batch size of 192 for the base AIM model, 160 for the BEV semantics variant, 72 for both the 2D semantics variant and 2D semantics + depth, and 24 for the control model of AIM-VA.

5. Additional Results

In this section, we provide more details regarding the Driving Score (DS) metric for evaluation on CARLA (Section 5.1) and show additional quantitative and qualitative experimental results. In particular, we provide a town-wise breakdown of the performance of all the baselines and NEAT (Section 5.2), discuss in more depth the results of the ablation study in the main paper (Section 5.3), and provide additional visualizations of the attention maps generated by NEAT (Section 5.4).

[9] https://github.com/open-mmlab/mmsegmentation

Method	Auxiliary Supervision	Town01	Town02	Town03	Town04	Town05	Town06
CILRS [7]	Velocity	24.59	33.73	25.62	24.53	3.56	28.11
LBC [2]	BEV Sem	32.03	32.38	25.85	27.89	34.80	23.70
AIM [19]	None	63.53	47.06	33.12	48.59	65.36	60.21
AIM-MT	2D Semantics	61.77	44.67	30.03	70.13	72.79	76.20
	BEV Semantics	63.12	47.96	34.70	77.40	51.16	93.88
	Depth+2D Semantics	59.79	46.01	45.17	81.09	64.94	93.89
AIM-VA	2D Semantics	71.82	51.73	36.82	76.88	54.54	77.97
NEAT	BEV Semantics	74.64	78.94	39.12	81.56	49.81	71.23

Table 2: **Town-wise Driving Score.** We show the mean of 3 evaluations for each model on the 42 evaluation routes.

5.1. Metrics

For CARLA, the driving proficiency of an agent is characterized by its Driving Score (DS). This metric can be broken down into two components (Route Completion and Infraction Score), which we describe below.

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

$$\text{RC} = \frac{1}{N}\sum_{i}^{N} R_i \tag{7}$$

2. **Infraction Score** (P_i): geometric series of infraction penalty coefficients, p_i^j for every instance i of infraction j incurred by the agent during the route. Agents start with an ideal 1.0 base score, which is reduced by a penalty coefficient for every infraction.

$$P_i = \prod_{j}^{\text{ped,....,stop}} (p_i^j)^{\#\text{ infractions}(j)} \tag{8}$$

Each infraction is pre-assigned a different base penalty p^j, which we list below.

- Collisions with pedestrians — 0.50.
- Collisions with other vehicles — 0.60.
- Collisions with static elements — 0.65.
- Running a red light — 0.70.
- Running a stop sign — 0.80.

Besides these, there is one additional infraction that has no explicit fixed coefficient. If an agent drives off-road, that percentage of the route will not be deducted with a multiplier $(1-\text{off road percentage})$. The **Driving Score (DS)** is the weighted average of the route completion with the infraction multiplier P_i. DS is the main metric used in both our experiments and for ranking models on the CARLA Leaderboard.

$$\text{DS} = \frac{1}{N}\sum_{i}^{N} R_i P_i \tag{9}$$

5.2. Town-wise results

The town-wise driving score of the methods compared in the main paper is shown in Table 2. CILRS and LBC perform poorly across all towns. For the other models, we observe that Town03 is consistently difficult, with even the best of all models having a driving score below 50. On Town02, which is very difficult for all the baselines due to its dense traffic relative to the small map size, NEAT outperforms the next best method by over 25 points.

Variant	Value	Driving Score ↑	Waypoint L1 ↓	Segmentation IoU ↑					
				Unlabeled	Obstacle	Road	Red Light	Green Light	Lane Marking
Training Seeds	1	67.04	0.242	0.805	0.798	0.585	0.688	0.534	-
(Default Configuration)	2	64.41	0.252	0.814	0.800	0.585	0.707	0.536	-
4 Classes (- Green Light)	$K = 4$	48.65	0.242	0.841	0.784	0.641	0.779	-	-
6 Classes (+ Road Line)	$K = 6$	57.86	0.276	0.780	0.750	0.377	0.738	0.581	0.579
Less Iterations	$N = 1$	46.13	0.249	0.816	0.827	0.602	0.717	0.554	-
More Iterations	$N = 3$	51.82	0.234	0.825	0.803	0.595	0.685	0.483	-
Shorter Horizon	$Z = 2$	38.65	0.098	0.825	0.849	0.651	0.793	0.636	-
Longer Horizon	$Z = 6$	50.54	0.440	0.779	0.811	0.545	0.673	0.515	-
No Side Views	$S = 1$	55.13	0.246	0.751	0.721	0.520	0.710	0.523	-
No Transformer	$L = 0$	62.93	0.263	0.804	0.792	0.578	0.644	0.468	-
No Intermediate Loss	$\gamma_1 = 0$	51.78	0.253	0.795	0.829	0.607	0.611	0.477	-
No Semantic Loss	$\lambda = 0$	36.29	0.287	-	-	-	-	-	-
No Red Light Indicator	$\mathbf{r} = 0$	47.62	0.242	0.805	0.798	0.585	0.688	0.534	-

Table 3: **Ablation study.** We show the mean DS over the 42 CARLA routes for a single evaluation run of different NEAT model configurations. In addition, we show the waypoint L_1 loss and class-wise segmentation IoUs for these configurations on our offline validation set. The default configuration of $K = 5$, $N = 2$, $L = 2$, $Z = 4$, $S = 3$, $L = 2$, and $\gamma_1 = 0.1$ with a red light indicator leads to the best driving scores.

5.3. Ablation Study

In Table 3, we compare 2 random seeds of our default model configuration to variants with different choices for several parameters. In addition to the Driving Score metric reported in the main paper obtained by closed-loop evaluation on our 42 CARLA routes, we show the offline performance of these models when evaluated on our validation dataset in terms of the L_1 loss between ground truth and predicted waypoints, as well as the class-wise segmentation IoU. The class-wise intersection and union scores for our IoU computation are obtained by sampling points using our class-balancing procedure during validation and comparing the predicted and ground truth classes for these points. Note that the online metric (DS) and offline metrics are not always correlated to each other, in line with previous findings in Imitation Learning literature [5].

Which semantic classes help most? In this experiment, we consider 3 supervised variants of our method with different classes used for BEV semantic prediction. We start with a 6-class configuration ($K = 6$) based on the efficacy of this set of classes shown in [1]: (1) unlabeled, (2) obstacle (person + vehicle), (3) road, (4) lane marking, (5) non-green light (red light + yellow light), (6) green light. The class combinations for person + vehicle and red light + yellow light are introduced for our model due to the lack of sufficient ground truth person and yellow light samples in our dataset. Further, we consider a 5-class subset ($K = 5$) by merging lane marking with road, which is our default configuration, and a 4-class subset ($K = 4$) by further merging green light with unlabeled. The idea behind these subsets is to check if simplifying the perception task is beneficial for driving. For example, without an explicit green light class, the network may infer the absence of a red light at an intersection to be equivalent to a green light. Our results are shown in (Table 3), where the default configuration of $K = 5$ performs best. While the drop in performance with fewer classes is intuitive, it is surprising that we also observe a drop for $K = 6$. This is because predicting accurate BEV lane markings is challenging due to their thin, elongated structure, and including this class reduces the quality of segmentation for other classes (specifically road, as evidenced by the IoU drop in Table 3). This is illustrated in Fig. 4.

In contrast to NEAT, the 2D semantic segmentation models in our study (LBC, AIM-MT, and AIM-VA) show significant drops in performance when using only the 5 classes of NEAT's default configuration. For example, the driving score of AIM-MT (2D depth + 2D semantics) drops from 64.86 to 51.79 when using NEAT's 5 classes.

How important is the iterative attention loop? In our default model configuration, we use an iterative process with $N = 2$ iterations. Here, we investigate a simpler, non-iterative architecture ($N = 1$), as well as the impact of having additional attention iterations ($N = 3$). As shown in Table 3, the simpler variant ($N = 1$) obtains good segmentation IoU scores, but is unable to compete with the default model ($N = 2$) in terms of driving. The performance also drops when increasing the number of attention iterations from the default to $N = 3$, which is possibly due to sub-optimal convergence because of the increased optimization complexity.

Is a longer prediction horizon Z useful? As shown in Table 3, NEAT's driving score drops significantly with a shorter prediction horizon ($Z = 2$). For this model, the output waypoints do not deviate sharply enough from the vertical axis for the

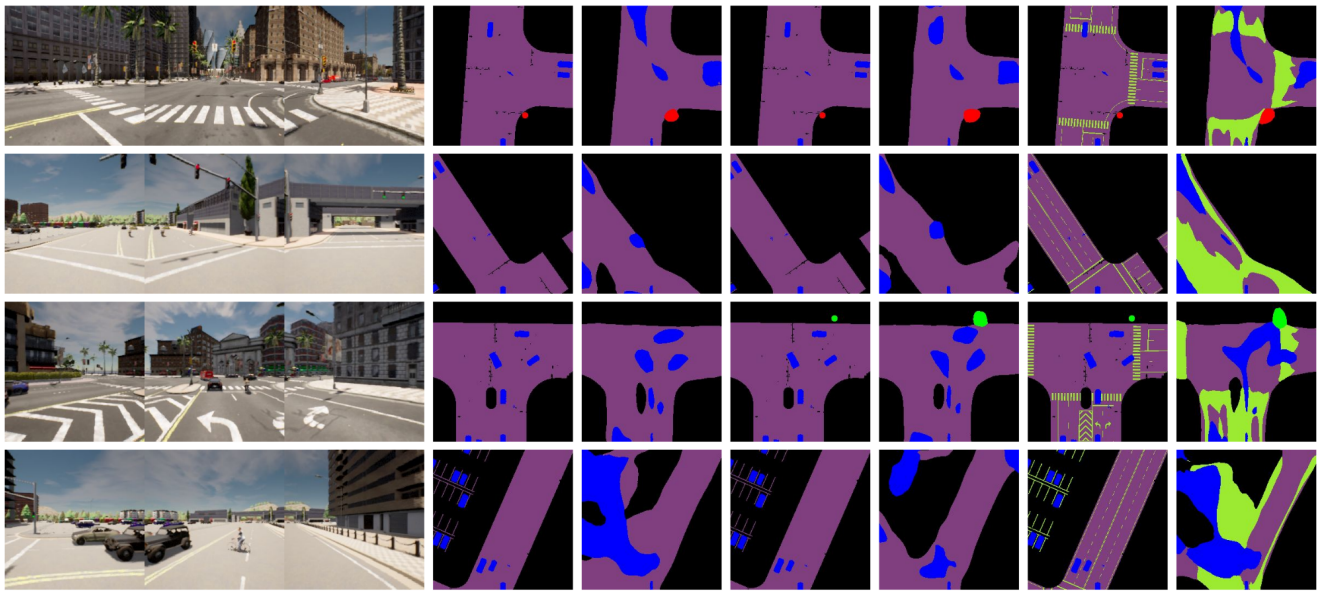

Figure 4: **Semantic class configurations.** We show the ground truth (GT) and predicted (Pred) semantics for models trained with 4, 5, and 6 semantic classes, for 4 different input scenes. Note the fine structure of the lane markings in GT (6), which leads to large errors in the Pred (6) visualizations in the areas of the scene containing lane markings.

controller to perform certain turning-related maneuvers. On the other hand, a longer horizon complicates the prediction task, with $Z = 6$ giving worse results than the default $Z = 4$. Note that for both these experiments, the offline validation metrics (Waypoint L1 and Segmentation IoU) are only included in Table 3 for reference, and are not directly comparable to models from the other ablation experiments. This is because predicting fewer waypoints and semantics only for short time horizons is a much simpler task and vice versa.

What is the impact of additional camera viewpoints? We now study the impact of changing the number of input sensors (S). Combining features from $S = 3$ views leads to improvements for our model, showing that NEAT is an effective way to obtain a unified scene representation from multiple sensors. In particular, we observe improvements in the Segmentation IoU for nearly all classes and a better overall driving score. The reduced segmentation IoU with $S = 1$ is likely because some of the points for which NEAT must predict semantics lie outside the field of view of a single camera. While we observed a similar trend (better performance with 3 input views) for the LBC baseline (which also uses BEV semantics during training), this is not the case for all methods. We obtained no improvements for CILRS, AIM (and its 2D multi-task variants), as well as AIM-VA when using three input views.

Is the transformer encoder essential? We compare the base model ($L = 2$) to a simpler architecture with no transformer ($L = 0$) in Table 3. Without the transformer, there is a small drop in performance due to the lack of global context, but it is not as significant as the parameters discussed in Table 3. This showcases the ability of the iterative NEAT attention loop to perform feature aggregation with sufficient global reasoning for the self-driving task.

Is the intermediate loss helpful? We see in Table 3 that without this loss term, the performance drops almost to the level of the model with $N = 1$, showing its importance in guiding the learning of the iterative attention process.

Is the auxiliary semantic supervision necessary? The semantic loss is the most crucial ingredient in NEAT's representation, as evidenced by the sharp drop in driving performance in Table 3 (over 30%) when this loss is removed.

Is the red light indicator beneficial? In this experiment, we use the model trained with the default parameters, but explicitly set the red light indicator $\mathbf{r}$ (which is input to the longitudinal PID controller) to zero for all frames at test time. Note that the offline metrics for this model match our default configuration since we use the same model weights and only modify the controller used for driving. This variant has a much lower driving score, showcasing the importance of red lights in the new evaluation setting. Besides the penalty applied to the infraction score for running a red light, the agent often ends up committing further infractions such as colliding with other vehicles in the intersection. The red light indicator leads to NEAT being extra cautious, stopping as soon as a red light is observed in the BEV segmentation map, often a few meters away from

the actual line of infraction where the expert would stop.

It is also possible to implement a red light indicator for the AIM-MT models that predict semantics. For the best AIM-MT model from the main paper (with auxiliary 2D depth and 2D semantics), we ran an additional evaluation where the brake is manually set to 1 if any of the predicted 2D semantic pixels are of the class red light. However, we observe no improvements for this version of the model, with the mean driving score over 3 evaluations dropping from 64.86 to 63.54.

Is attention useful for the semantic prediction task? Table 3 shows the mIoU of different NEAT configurations, however, these metrics only consider $M = 64$ ground truth points per image, sampled as per the strategy described in Section 3.2 of the main paper. We now evaluate a baseline that encodes the 3 images, concatenates latent vectors, and decodes them to 256×256 BEV semantics with a broadcast decoder. We evaluate the mIoU over all 256^2 points. This baseline achieves a mIoU of 27.06%. In particular, its IoU on both "red light" (which has 171k pixels, 0.01% of the dataset) and "green light" (56k pixels, 0.004% of the dataset) is **zero**. In comparison, NEAT trained exclusively for segmentation (no waypoint loss) obtains an mIoU of 45.20% (with 13.40% on "red light" and 15.60% on "green light"). This shows that attention plays an important role in the semantic branch.

Is averaging the waypoint predictions important? Randomly sampling a single query out of the G^2 predictions (DS = 64.70) does not lead to a large drop in performance compared to taking the grid average (DS = 67.04), so it is not an integral component of our approach. However, qualitatively, the trajectories are sometimes less smooth when sampling rather than averaging, which is not penalized by the DS metric.

5.4. Attention Visualizations

Our supplementary video contains qualitative examples of NEAT's driving capabilities. For the first route in the video, we visualize attention maps $\mathbf{a}_N$ for different locations on the route. For each frame in the video, we randomly sample BEV (x, y) locations and pass them through the trained NEAT model until one of the locations corresponds to the class obstacle, red light, or green light. Four such frames are shown in the main paper. Twelve additional frames from the same route are shown here in Fig. 5.

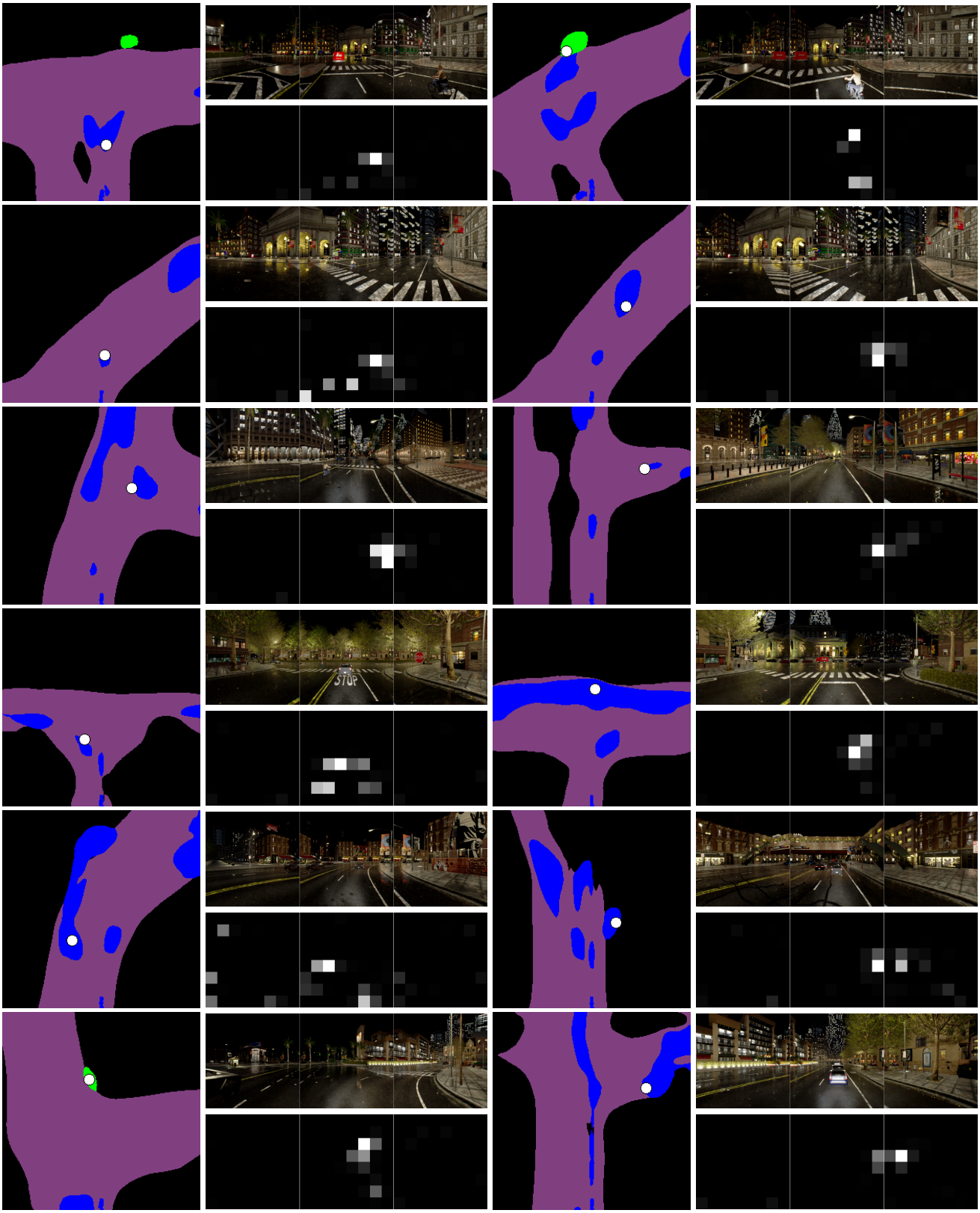

Figure 5: **Attention maps.** We visualize the semantics $\mathbf{s}_N$ for 12 frames of a driving sequence from Town10 in CARLA. We highlight one particular (x, y) value as a white circle on each $\mathbf{s}_N$, for which we visualize the multi-view input and corresponding attention map $\mathbf{a}_n$. NEAT consistently attends to the image region corresponding to the object of interest.

References

[1] Aseem Behl, Kashyap Chitta, Aditya Prakash, Eshed Ohn-Bar, and Andreas Geiger. Label efficient visual abstractions for autonomous driving. In *Proc. IEEE International Conf. on Intelligent Robots and Systems (IROS)*, 2020. 6, 9, 11

[2] Dian Chen, Brady Zhou, Vladlen Koltun, and Philipp Krähenbühl. Learning by cheating. In *Proc. Conf. on Robot Learning (CoRL)*, 2019. 4, 6, 10

[3] Mark Chen, A. Radford, Jeff Wu, Heewoo Jun, Prafulla Dhariwal, David Luan, and Ilya Sutskever. Generative pretraining from pixels. In *Proc. of the International Conf. on Machine learning (ICML)*, 2020. 1

[4] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua 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. 8

[5] Felipe Codevilla, Antonio M. Lopez, Vladlen Koltun, and Alexey Dosovitskiy. On offline evaluation of vision-based driving models. In *Proc. of the European Conf. on Computer Vision (ECCV)*, 2018. 11

[6] Felipe Codevilla, Matthias Miiller, Antonio López, Vladlen Koltun, and Alexey Dosovitskiy. End-to-end driving via conditional imitation learning. In *Proc. IEEE International Conf. on Robotics and Automation (ICRA)*, 2018. 7

[7] Felipe Codevilla, Eder Santana, Antonio M. López, and Adrien Gaidon. Exploring the limitations of behavior cloning for autonomous driving. In *Proc. of the IEEE International Conf. on Computer Vision (ICCV)*, 2019. 6, 7, 10

[8] Harm de Vries, Florian Strub, Jérémie Mary, Hugo Larochelle, Olivier Pietquin, and Aaron C. Courville. Modulating early visual processing by language. In *Advances in Neural Information Processing Systems (NIPS)*, 2017. 2

[9] Jia Deng, Wei Dong, Richard Socher, Li jia Li, Kai Li, and Li Fei-fei. Imagenet: A large-scale hierarchical image database. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2009. 7, 9

[10] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv.org*, 2010.11929, 2020. 1

[11] Vincent Dumoulin, Ishmael Belghazi, Ben Poole, Olivier Mastropietro, Alex Lamb, Martin Arjovsky, and Aaron Courville. Adversarially learned inference. In *Proc. of the International Conf. on Learning Representations (ICLR)*, 2017. 2

[12] Angelos Filos, Panagiotis Tigas, Rowan McAllister, Nicholas Rhinehart, Sergey Levine, and Yarin Gal. Can autonomous vehicles identify, recover from, and adapt to distribution shifts? In *Proc. of the International Conf. on Machine learning (ICML)*, 2020. 8

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2016. 2

[14] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proc. of the International Conf. on Learning Representations (ICLR)*, 2015. 9

[15] Peilun Li, Xiaodan Liang, Daoyuan Jia, and Eric P. Xing. Semantic-aware grad-gan for virtual-to-real urban scene adaption. *arXiv.org*, 1801.01726, 2018. 8

[16] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *Proc. of the International Conf. on Learning Representations (ICLR)*, 2019. 9

[17] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems (NIPS)*, 2019. 2, 9

[18] Aditya Prakash, Aseem Behl, Eshed Ohn-Bar, Kashyap Chitta, and Andreas Geiger. Exploring data aggregation in policy learning for vision-based urban autonomous driving. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2020. 5

[19] 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. 3, 6, 8, 10

[20] Di Qi, L. Su, Jia Song, E. Cui, Taroon Bharti, and Arun Sacheti. Imagebert: Cross-modal pre-training with large-scale weak-supervised image-text data. *arXiv.org*, 2001.07966, 2020. 1

[21] Stéphane Ross, Geoffrey J. Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Conference on Artificial Intelligence and Statistics (AISTATS)*, 2011. 5

[22] Chen Sun, Austin Myers, Carl Vondrick, Kevin Murphy, and Cordelia Schmid. Videobert: A joint model for video and language representation learning. In *Proc. of the IEEE International Conf. on Computer Vision (ICCV)*, 2019. 1

[23] 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. 1

[24] Nicholas Watters, Loïc Matthey, Christopher P. Burgess, and Alexander Lerchner. Spatial broadcast decoder: A simple architecture for learning disentangled representations in vaes. In *Proc. of the International Conf. on Learning Representations (ICLR) Workshops*, 2019. 9