CaRL: Learning Scalable Planning Policies with Simple Rewards

Bernhard Jaeger **Daniel Dauner** **Jens Beißwenger**
Simon Gerstenecker **Kashyap Chitta*** **Andreas Geiger**
University of Tübingen, Tübingen AI Center
{bernhard.jaeger, daniel.dauner, a.geiger}@uni-tuebingen.de

Abstract: We investigate reinforcement learning (RL) for privileged planning in autonomous driving. State-of-the-art approaches for this task are rule-based, but these methods do not scale to the long tail. RL, on the other hand, is scalable and does not suffer from compounding errors like imitation learning. Contemporary RL approaches for driving use complex shaped rewards that sum multiple individual rewards, e.g. progress, position, or orientation rewards. We show that PPO fails to optimize a popular version of these rewards when the mini-batch size is increased, which limits the scalability of these approaches. Instead, we propose a new reward design based primarily on optimizing a single intuitive reward term: route completion. Infractions are penalized by terminating the episode or multiplicatively reducing route completion. We find that PPO scales well with higher mini-batch sizes when trained with our simple reward, even improving performance. Training with large mini-batch sizes enables efficient scaling via distributed data parallelism. We scale PPO to 300M samples in CARLA and 500M samples in nuPlan with a single 8-GPU node. The resulting model achieves 64 DS on the CARLA longest6 v2 benchmark, outperforming other RL methods with more complex rewards by a large margin. Requiring only minimal adaptations from its use in CARLA, the same method is the best learning-based approach on nuPlan. It scores 91.3 in non-reactive and 90.6 in reactive traffic on the Val14 benchmark while being an order of magnitude faster than prior work.

Keywords: Autonomous Driving, Reinforcement Learning, Planning

1 Introduction

We consider the task of privileged planning, in which an autonomous vehicle drives using ground truth perception inputs. Such planners are traditionally rule-based [1, 2, 3, 4, 5]. While rule-based approaches work well for regular driving [4], they require special scenario-specific rules to solve more complex scenarios [5], which is unlikely to scale to the long tail of driving scenarios.

Training neural planners with imitation learning (IL) is a popular alternative to rule-based approaches [6, 7, 8, 9, 10, 11, 12, 13], because these methods can scale with data. Yet surprisingly, these methods underperform compared to rule-based or hybrid approaches [4]. A common explanation for this behavior is that IL suffers from a distribution shift between the open-loop training objective and the closed-loop inference task.

Closed-loop training [15], in particular Reinforcement Learning (RL) [16, 17], is a promising alternative. A key problem in RL for driving is designing an appropriate reward function [18]. Early work in RL for driving used principled, simple rewards [19], such as maximizing forward speed and terminating upon infractions. Empirically, however, learning with such simple rewards only succeeded in environments without other actors [19, 20, 21, 22], where simple behaviors suffice.

*Work done at the University of Tübingen, Kashyap Chitta is currently affiliated with NVIDIA Research.

9th Conference on Robot Learning (CoRL 2025), Seoul, Korea.

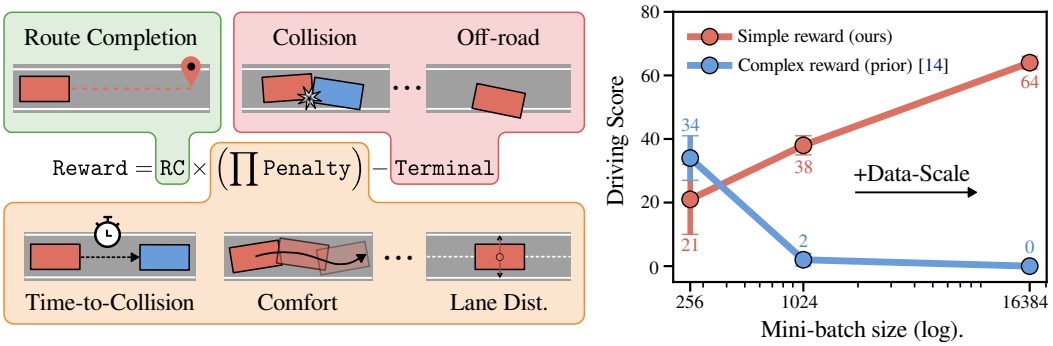


Figure 1: **Simple rewards scale with mini-batch size.** Typical rewards in driving consist of complex rewards that trade off many individual components. This limits scalability as PPO gets stuck in local minima with larger mini-batch sizes. We propose a simple alternative based on maximizing route completion that scales well with mini-batch size.

In settings where other dynamic actors are present, which require more complex behavior, such as reacting to sudden changes in the environment, simple rewards were empirically found to provide insufficient supervision [23]. As a result, recent designs provide denser feedback to simplify learning, combining many rewards additively, such as one reward for speed, orientation, position, and comfort [18]. The downsides of this approach are that the tradeoffs between rewards need to be carefully tuned, local minima are introduced, and the agent exploits undesirable shortcuts. These tradeoffs harm scalability: For example, we observe that increasing the mini-batch size by a factor of 4 with a complex reward reduces the performance of Proximal Policy Optimization (PPO) [24] drastically, due to a local optimum of the reward. Larger mini-batch sizes smooth the gradient, which can make optimization more prone to local optima. Additionally, many popular rewards [25, 14, 15, 26, 27] rely on simplistic rule-based planners to compute their reward terms. This upper bounds the performance, as the decisions from the rules are rewarded as if they were optimal.

We propose an alternative reward design that does not rely on rule-based planners. The design learns policies with route completion (RC) as the only source of reward. To learn to avoid infractions, we end the episode upon any major infraction, e.g. collision, and reduce the obtained route completion multiplicatively while the agent is violating soft constraints, e.g., exceeding the speed limit.

While conceptually appealing, prior empirical findings suggest that such a simple reward does not provide enough feedback for the policy to learn effectively. We reproduce the most popular CARLA [28] RL planner Roach [14] on the CARLA leaderboard 2.0 [29] and show that it naively performs worse when trained with the simpler reward. However, unlike the complex reward, increasing the mini-batch size from 256 to 1024 drastically improves learning performance, with the simple reward outperforming the complex Roach reward. This is illustrated in Fig. 1.

Large mini-batch sizes, made possible by our reward, enable training with much more data, as data collection can be efficiently parallelized. We scale our training to 300 million samples on CARLA, 30x more than prior work, with one compute server and mini-batch size of 16384. This results in a massive performance improvement.

Our final model, named CaRL, outperforms Roach and the recent world model RL-planner Think2Drive [27] on the longest6 v2 [30] benchmark, by 42 and 57 Driving Score (DS) respectively. We also implement our method on the nuPlan [31] simulator, which measures performance in realistic everyday scenarios via log replay. We show that with minimal changes, our method can achieve 91 closed-loop score on the Val14 benchmark in both non-reactive and reactive traffic. The resulting model is both 1.7 (non-reactive) and 7.9 (reactive) points better than the prior best learning based approach, Diffusion Planner [32], while being 10x faster at inference time.

Compared to other approaches, there has been little RL research in planning. One reason might be that there is no publicly available RL code base for both the CARLA and nuPlan simulators. To foster reproducible research, we published our code at https://github.com/autonomousvision/CaRL.

2 Improving Scalability for RL

Implementing RL on the CARLA leaderboard 2.0 is challenging because the simulator is slow [27]. In this section, we hence propose better hyperparameters and an optimized code base for RL in CARLA. These changes make it possible to train on-policy RL methods on the CARLA leaderboard 2.0 efficiently. Related work is discussed in Appendix Section A.

2.1 Preliminaries

We investigate the task of urban navigation from point A to B [30] with ground truth perception. In sections 2 and 3, we develop our method on CARLA [28, 29] with a set of ablations where hyperparameters are held constant. In Section 4, we perform system-level comparisons with prior work and additionally evaluate our method on the nuPlan [31] simulator.

Benchmark: We use the CARLA simulator version 0.9.15 and the longest6 v2 benchmark. Longest6 v2 consists of 36 routes in towns 1-6 that are between 1 and 2 km long. Along the route, between 5 and 21 pre-crash safety-critical scenarios [33] of 7 different types, as defined by the CARLA leaderboard 2.0, appear. Longest6 [30] is a popular CARLA leaderboard 1.0 benchmark. We converted the scenario definitions using the official scenario converter [34] to the leaderboard 2.0 style and use no modification to the leaderboard code. We name the resulting benchmark longest6 v2 to avoid confusion, since results on longest6 v2 are not comparable to results on longest6. The background traffic can drive up to 80 km/h in v2, 2-3 times faster than in leaderboard 1.0, making the benchmark significantly harder. We focus on this benchmark because it uses the towns 1-6, which run substantially faster than the huge towns 12 and 13, allowing for faster RL and experimentation. **Metrics:** We use the standard CARLA leaderboard 2.0 metrics Route Completion (RC), and Driving Score (DS). DS multiplies RC with a penalty that decreases for every infraction. Details of the metrics can be found in [35]. **Algorithm:** We train our model with Proximal Policy Optimization (PPO) [24]. PPO is one of the most popular RL algorithms due to its stable convergence and high asymptotic performance. **Architecture:** We use bird's eye view semantic segmentation as input and predict the action of the car with a small CNN. **Variance:** To combat the variance of CARLA evaluations [36, 37, 30] and RL training [38, 39, 40, 41] every experiment averages 5 training seeds, each evaluated 3 times, unless noted otherwise. We report standard deviation across training seeds.

Simulation speed: Our method builds upon Roach [14], the best open-source RL method trained with PPO on CARLA. Roach has previously not been trained on the CARLA leaderboard 2.0. We first reproduce Roach on the CARLA leaderboard 2.0 and then carefully modify it. As already documented in [27], the CARLA leaderboard 2.0 is slow for RL because of its slow reset step (up to 60 seconds). To address this problem, we created a separate CARLA leaderboard 2.0, which we optimized for RL training. We use the original leaderboard 2.0 code for evaluation to ensure fair comparisons. The full list of optimizations can be found in the appendix and code. Noteworthy optimizations include avoiding town reloading by specializing each simulator instance to a particular town and pre-processing the A* route planning. Due to these optimizations, we were able to reproduce Roach with 10 million environment samples in 32 hours on a single A100 GPU.

2.2 Enabling higher learning rates

Selecting a robust base set of hyperparameters for PPO is important in planning because the slow simulators and long training times prevent automatic hyperparameter tuning [42]. In this section, we show that the PPO Atari hyperparameter set [24, 43] enables us to train better models faster, compared to the standard Roach hyperparameters used in CARLA [14], reducing the computational cost of our experiments. One advantage of the Atari hyperparameters is that they enable training at higher learning rates. This effect can be intuitively explained by examining the number of off-policy steps performed by PPO.

Off-policy steps: Policy gradient methods like PPO estimate the gradient of the policy's performance and perform gradient ascent. It is possible to get an unbiased estimate of this gradient for on-policy learning, i.e., if only one optimizer step is used before the data is discarded and new data recollected. Strict on-policy methods like REINFORCE [44] or A2C [45] do this, but are sample inefficient. PPO instead increases data efficiency by performing multiple gradient steps per iteration, approximating the off-policy policy gradient [46]. The resulting approximation error is controlled by limiting policy change via the clipping heuristic. Another heuristic is that the policy is only updated for a few gradient steps, limiting policy change implicitly. Policies typically do not change a lot per gradient step, although this is dependent on the learning rate.

Table 1 compares the number of off-policy steps between the standard CARLA parameters from Roach [14] and the Atari [24, 43] parameters.

Hyperparameter	**Atari** [24, 43]	**CARLA** [14]
Initial Learning rate	0.00025	0.00001
Steps off-policy (steps$\times$epochs-1)	$(4 \times 4 - 1) = 15$	$(48 \times 20 - 1) = 959$

Table 1: **Default Hyperparameters of PPO for CARLA and Atari.**

Roach performs 959 off-policy steps (the first step is on-policy), 60 times more than the Atari parameters. One would expect that this introduces large policy changes and hence off-policy error, and it is surprising that Roach converges at all. The reason for this is the small learning rate, which is 25x lower than the Atari learning rate, leading to smaller policy changes, hence balancing this effect. Training Roach with the (higher) Atari learning rate and schedule, without changing other hyperparameters, leads to a degenerate policy resulting in a drop of DS from 22 ± 14 to 2 ± 1 DS.

We train Roach with its hyperparameters and the Atari hyperparameters. The results are shown in Table 2. Surprisingly, we observe a 10-hour reduction in training time while the policy achieves 11 DS better performance. The reduction in training time comes from using 5 times fewer epochs, whereas the performance improvement might come from a combination of hyperparameters. We therefore use the Atari hyperparameters as the basis for our model.

Hyperparameter set	**Train Time h** $\downarrow$	**Output**	**DS** $\uparrow$	**RC** $\uparrow$
CARLA [14]	32 $\pm$ 2	mode [14]	22 $\pm$ 14	77 $\pm$ 19
Atari [24, 43]	**22** $\pm$ 1	mode [14]	33 $\pm$ 10	**90** $\pm$ 6
		mean	**34** $\pm$ 7	86 $\pm$ 6

Table 2: **Atari vs CARLA hyperparameter set on CARLA.** The Atari hyperparameters train faster (-10 h using one A100 (40GB) and 14 EPYC 7742 CPU cores), achieve better performance (+11 DS), and have lower variance (-4 std).

Roach uses a Beta distribution to parametrize its action space. During inference, sampling from the distribution is typically replaced with a statistic of the distribution to turn off exploration. Roach proposes to use the mode of the distribution when $\alpha, \beta > 1$. Using the mode biases the policy's behavior, as PPO optimizes the expected performance under the action distribution (the mean). We observe that using the mean achieves 1 DS better performance with the same models, and reduces the standard deviation by 3 points.

3 Optimizing a Single Reward

Many current RL planners [25, 14, 26, 27] use a reward function that consists of multiple additive terms, including terms to reward the inverse distance to the desired vehicle speed, desired vehicle position, and desired orientation. This is reminiscent of imitation learning because these "desired" states are computed using handcrafted rules akin to a rule-based planning method, or even directly

with human labels [47]. As noted in the Appendix of [14], their rule-based planner is suboptimal and sometimes assigns higher rewards to suboptimal states. The advantage of these complex, shaped rewards is that they can simplify learning. They provide dense, rich feedback, which simplifies the credit assignment problem of RL.

These rewards have, however, many downsides. The handcrafted rules, being suboptimal, can upper-bound performance, and these rewards introduce loopholes and local minima that the optimization can get stuck in, or the agent can exploit. One example of a failure case we observe with the Roach reward [14] is that the policy sometimes learns to wait at green traffic lights. This is a simple behavior that obtains a large return with the Roach reward. The model is constantly rewarded by the optimal speed reward if it has speed 0 at a red light. Because traffic lights are often red for a long time and green for a short amount of time in CARLA, the policy sometimes learns to exploit that it can just wait for the next red light for easy rewards, when the light turns green for a short duration. This problem is illustrated in Appendix Fig. 5. Eventually, many training seeds escape this local minima, but we still observe that the final policy drives slower than usual when approaching green lights, presumably trying to catch the next red light. This is illustrated in Appendix Fig. 6.

To address these concerns, we are proposing a new type of reward design:

$$r_t = RC_t * \left(\prod p_t\right) - T \tag{1}$$

The reward at time step t is computed by multiplying the percentage of the route completed during this simulator time step RC_t with soft penalty factors $p_t \in [0, 1]$.

The soft penalties p_t are 1 if their condition is not violated and otherwise have a value $\in [0, 1)$ depending on the type of infraction. Soft penalties are constraints that the agent should typically adhere to, such as staying within the speed limit, but may violate in order to avoid a hard penalty like a collision. It is important that soft penalty factors that the agent cannot avoid violating early on in training, such as comfort, need to be > 0. Otherwise, the agent would receive no reward. It is possible to apply a soft penalty for multiple frames e.g. to penalize actions while the car is not moving or increase the penalty strength.

Any major infraction, like a collision, is treated as a hard penalty. Hard penalties simply end the episode. The principle is that any hard constraints of the optimization problem are formulated as terminal states, ensuring that violating them results in suboptimal performance as no further reward can be collected. T is a terminal penalty that is applied at the end of the episode, depending on the infraction. It can be used to induce an ordering between infractions, but it needs to be sufficiently small that the agent is not penalized for starting to drive early on in training. We use $T = 1$ for collisions and red light infractions, and $T = 0$ for everything else. The full description of penalties we use can be found in Appendix Section F.

The reward design follows the following principles: (1) *The amount of reward obtainable is finite.* There is only 100 RC to be collected in total, preventing infinite reward loopholes like the car waiting at a green light. (2) *The reward only specifies what to do, not how to do it.* Our reward does not include any rule-based planners. (3) *The global optimum of the reward is the same as the global optimum of the metric* [17]. This ensures that solving the learning problem results in an optimal policy, wrt. the metric. Our reward closely aligns with the DS metric [48], with some additional penalties like speeding. The agent obtains the optimal reward when completing the route without infraction, which also yields the optimal DS.

Our reward is simple and gives much fewer local hints to the policy than other rewards. It avoids the problems of typical reward designs, but might be harder to optimize. Table 3 shows that PPO struggles to perform well without the rule-based dense supervision of the Roach reward, dropping performance by 13 points at the standard mini-batch size of 256. Interestingly, this trend reverses when we scale the mini-batch size to 1024. When training PPO with the dense Roach reward at 1024 mini-batch size, we observe that the model gets stuck in the local minimum of not driving on many routes, which yields good position and orientation rewards but suboptimal speed rewards. At the end

Reward	Local Hints?	Mini-Batch Size 256	Mini-Batch Size 1024
Roach [14]	✓	34 ± 7	2 ± 2
CaRL (Ours)	✗	21 ± 11	**38** ± 3

Table 3: **Reward and mini-batch size.** The metric is DS. PPO gets stuck in a local minimum when optimizing a complex reward at large mini-batch sizes. The simple reward does not have this problem. PPO even improves with a larger mini-batch size.

of training, the model started to learn to accelerate but hasn't learned to steer yet, leading to a poor driving policy with 2 DS. One hypothesis is that larger mini-batch sizes smooth the optimization, making it more prone to local minima. In contrast, our reward does not have these tradeoffs between different reward components, and PPO improves drastically when increasing the mini-batch size to 1024, improving DS by 17 points. Additionally, our reward outperforms the roach reward by 4 points in DS and reduces training variance from ± 7 to ± 3.

3.1 Scaling Data

There are two ways to increase mini-batch size in PPO while keeping the number of gradient steps per epoch unchanged. In the previous section, we increased the mini-batch size by performing 4x more simulator steps per PPO iteration with 4x fewer PPO iterations, because this method has similar resource requirements. If more resources are available, the number of parallel simulators can instead be increased, which raises the overall sample throughput. This also requires additional GPUs for model inference and training, which we utilize via the DD-PPO [20] scaling approach without preemption for the experiment in this section. DD-PPO works like distributed data parallelism [49] from supervised learning. Prior works trained PPO for driving with 1-10 million samples [14, 27]. This is typical for classic PPO [24], but 10-100 times less than what recent breakthroughs in robotics used [50, 22]. Prior work may not have scaled up data because increasing mini-batch size, which is required to scale efficiently, can yield degenerate performance, as Table 3 showed. Our reward enables us to scale our model from 10 million to 300 million samples using a single node with 8 A100 (40G) GPUs and 108 EPYC Rome CPU cores for 1 week. Table 4 shows that increasing the samples and mini-batch size leads to a large improvement of 33 DS. The baseline features some changes compared to the model in the last section, which are discussed in Appendix Section B. The 300M result is an average of three seeds.

Samples	Mini-batch	DS ↑	RC ↑
10M	1024	31 ± 7	63 ± 8
300M	16384	**64** ± 2	**82** ± 1

Table 4: **Effect of scale.**

4 Experiments

This section shows the advantages of our method over several other planning approaches on the CARLA longest6 v2 [30] benchmark and nuPlan Val14 [4]. All experiments were run by us with the same setting to ensure a fair comparison and avoid the common benchmarking errors that permeate the literature [51].

4.1 CARLA

Benchmark: We use the longest6 v2 benchmark, and metrics as described in Section 2.1.

Baselines: (1) **Roach** [14] is a popular RL planner trained with the PPO algorithm. (2) **Think2Drive** [27] is a recent RL planner that uses the world model-based approach DreamerV3 [52]. We reproduce both Roach and Think2Drive based on the details provided in the respective publications. (3) **PDM-Lite** [5] is a rule-based planning method in CARLA. The planners considered in this work are privileged, meaning they have access to ground truth perception inputs. These

Method	Type	DS ↑	RC ↑	Ped ↓	Veh ↓	MS ↓	Time ↓
PDM-Lite [5]	Rule	**73**	**100**	**0.00**	**0.18**	**7.67**	**18**
PlanT [6]	IL	62 ± 2	**96** ± 2	0.07	0.43	3.18	18
Think2Drive [27]	RL	7 ± 1	39 ± 19	0.97	2.55	18.05	**6**
Roach [14]	RL	22 ± 14	77 ± 19	0.45	2.42	7.12	7
CaRL (Ours)	RL	**64** ± 2	82 ± 1	**0.01**	**0.36**	**1.71**	8

Table 5: **Performance on longest6 v2** (CARLA).

inputs could realistically be predicted by a perception stack, albeit at lower accuracy. PDM-Lite is special in that it extracts additional information about the scenarios from the CARLA leaderboard that is not predicted by any existing perception stack. PDM-Lite knows which scenario type will appear and what the scenario parameters are, and uses this information to solve some of the scenarios with specialized rules that are specific to the scenario. Such an approach works for the small variety of scenarios encountered in CARLA, but is unrealistic to scale to the long tail. PDM-Lite is therefore perhaps better viewed as an auto-labeling method for imitation planners. (4) **PlanT** [6] is the SotA imitation planner for the CARLA leaderboard 1.0. It uses bounding boxes as input, which are processed with a transformer [53] and predicts waypoints. We reproduce PlanT for the CARLA leaderboard 2.0 by training it to imitate PDM-Lite. Implementation details about the baselines can be found in Appendix Section E.

Results: Our method, CaRL, achieves 64 DS on longest6 v2 as shown in Table 5. It significantly advances the SotA in RL planning, outperforming the best prior method, Roach, by 42 DS. It particularly reduces pedestrian (Ped) and vehicle (Veh) collisions compared to other RL baselines. The recent world model-based Think2Drive method only achieves a DS of 7 and is even outperformed by the Roach method when both methods are trained with the hyperparameters proposed by the respective papers. CaRL is the best learning based method outperforming PlanT by 2 DS while using a smaller model and less inference compute. The rule-based method PDM-Lite still achieves the best DS with 73 in particular due to its high route completion and low collisions. PDM-Lite outperforms CaRL because it is more consistent at solving some of the safety-critical scenarios. PDM-Lite drives relatively slow on average, which may help it avoid collisions but as a result incurs many min-speed infractions (MS). CaRL drives 31% faster than PDM-Lite on average (16.4 km/h vs 12.5 km/h), which is indicated by its 4.5 times lower MS infraction.

We additionally report the average runtime per time step in milliseconds (ms), including preprocessing observations, on the last route of longest6 v2. Times are measured with an `RTX 3090` GPU and an `i9-10850K` CPU. We observe that the RL-based methods are more efficient than the rule-based and imitation-based methods. This is because the model size of RL policies is typically smaller than models used in imitation learning. CaRL runs 2 times faster than the state-of-the-art approach PDM-Lite. It also uses a similar compute budget at inference as other RL-based approaches while driving substantially better.

4.2 nuPlan

Benchmark: NuPlan is a closed-loop simulator using real-world data. As metric, we use the closed-loop score (CLS), which is a weighted average of progress, time-to-collision, speed-limit compliance, and comfort, scaled in 0-100. Additionally, the score is set to zero if any hard penalty is violated e.g., collisions. We evaluate all methods using non-reactive log replay (NR) and reactive background traffic (R), where other vehicles are controlled with the IDM [1]. Following [4], we use the Val14 benchmark that includes 1118 simulations from the validation split.

Baselines: (1) **PDM-Closed** [4] is an extension of IDM [1] with multiple trajectory proposals, internal simulation, and scoring. (2) **PLUTO** [12] outputs multiple learned trajectories and scores,

Method	Type	Non-Reactive			Reactive			Time ↓
		CLS ↑	Col. ↑	RC ↑	CLS ↑	Col. ↑	RC ↑	
Log Replay (LQR)	*Human*	*93.5*	*98.8*	*99.0*	*80.3*	*85.6*	*99.0*	-
PDM-Closed [4]	Rule	**92.8**	**98.1**	92.1	**92.1**	**97.9**	**90.3**	**104**
PLUTO [12]	IL+Rule	92.6	97.9	**93.0**	89.7	97.1	86.1	237
PlanTF [8]	IL	84.6	94.2	90.7	76.1	95.2	77.2	107
Diff. Planner [32]	IL	89.6	95.9	94.2	82.7	93.1	85.9	138
CaRL (Ours)	RL	**91.3**	**97.4**	**94.4**	**90.6**	**97.1**	**91.3**	**14**

Table 6: **Performance on Val14** (nuPlan)

which can be combined with the scoring mechanism of PDM-Closed. (3) **PlanTF** [8] vectorizes agents and map elements and processes them with a transformer to forecast non-ego agents, and regress a trajectory. (4) **Diffusion Planner** [32] applies a diffusion transformer to generate the ego trajectory conditioned on a vectorized scene representation. (5) **Log Replay** tracks the human trajectory with an LQR controller [54].

Adaptation: We use different baselines for CARLA and nuPlan because the code of these planners is only compatible with either CARLA or nuPlan. Prior work only evaluated on one of the two simulators. For a more rigorous evaluation, we reproduce CaRL on nuPlan as well. This requires one special adaptation. The simulation duration in nuPlan is a constant 15 seconds, even when the ego agent has already completed the route. Since we primarily reward route completion, the agent would get no reward signal after completing the route. To account for this simulator detail, we additionally reward the agent with a constant survival bonus at every frame during training, which gives the agent an incentive to avoid infractions after completing the route. We train CaRL with 500M samples on nuPlan since the simulation is faster. We refer the reader to Appendix Section B.6 for further implementation details.

Results: As shown in Table 6, rule-based methods are highly effective in nuPlan, with PDM-Closed achieving over 92 reactive and non-reactive CLS. IL planners achieve reasonable performance (85-90) in non-reactive traffic but have difficulties adapting from the non-reactive setting (as seen during training) to the reactive IDM traffic. E.g., Diffusion Planner (without post-processing) achieves a strong non-reactive CLS of 90 but a weaker reactive score of 83. PLUTO and PlanTF have similar performance drops. CaRL works well in both settings, achieving the highest CLS of all learned planners with 91.3 in the non-reactive and 90.6 CLS in reactive mode, respectively. CaRL runs $7 - 17\times$ faster than the baselines, due to its small 2M parameter network. We measure runtimes in ms on an `A5000` with an `i9-13900K`. The inference time of CaRL is slightly larger on nuPlan than on CARLA because nuPlan has more dynamic actors, which increases rendering times of the observation. We provide further results and baselines in Appendix E.2.

5 Conclusion

Contemporary RL methods for driving often use complex rewards that induce tradeoffs between multiple reward terms. We observe that these tradeoffs prevent scalability. Training with large mini-batch sizes with PPO leads to degenerate policies, likely since the optimization gets stuck in a local minimum of the reward. We propose an alternative reward design based on optimizing a single reward: route completion. Infractions either terminate the episode or multiplicatively reduce route completion. We show that this enables learning with large mini-batch sizes, which in turn enables efficient scaling to more samples by parallelizing data collection. We scale PPO to 300M samples in CARLA using a mini-batch size of 16384 and show that this leads to a performance improvement of 33 DS on the longest6 v2 benchmark. CaRL achieves 91 CLS on nuPlan Val14 in both non-reactive and reactive traffic, outperforming all prior learning-based approaches.

Limitations: We investigate urban driving at moderate speeds of up to 80 km/h. Problems specific to high-speed driving on highways are not considered.

Our work improves the SOTA of RL for driving in simulation. For RL to become relevant for real cars, Sim2Real transfer [55, 50, 22, 56] needs to be demonstrated, which we leave for future work.

We train CaRL with the 7 scenario types of longest6 v2. The CARLA leaderboard 2.0 offers more scenario types, which we have not investigated in this work.

For the CARLA vehicle physics, there are no human reference values for comfortable driving. Comfort is not a physical quantity but a subjective human feeling, so bounds need to be set based on human data. We set wide bounds to ensure that the model can comply with the comfort infraction, but the resulting behavior might not be comfortable in a real car. Future work may tune these bounds for smoother driving in CARLA.

CaRL has two main failure modes in CARLA: missing exits in highway off-ramps and other cars crashing into its rear in scenarios where another car runs a red light (rear-end collisions are counted as the agent's fault in CARLA). We show examples in the Appendix Section C.

Our reward does not encode that getting to the goal faster is better. This is because most RL algorithms naturally encode a notion of urgency via a discount factor.

Both Longest6 v2 and Val14 are level 4 training benchmarks, where training on the evaluation town is allowed. We have not investigated level 5 generalization to new towns.

Our reward gives fewer local hints than other rewards. We have deliberately chosen an RL algorithm (PPO) that uses Monte-Carlo returns for optimization, which sum up rewards. This might alleviate the absence of locality. RL algorithms based on Q-learning [57, 58], that rely on local TD-prediction, such as Soft-Actor Critic [59], may have a harder time optimizing our reward, although we have not investigated other algorithms.

We think these limitations can be addressed and are promising directions for future work.

Acknowledgments

Bernhard Jaeger and Andreas Geiger were supported by the ERC Starting Grant LEGO-3D (850533), the DFG EXC number 2064/1 - project number 390727645, and the Vector Stiftung. Daniel Dauner was supported by the German Federal Ministry for Economic Affairs and Climate Action within the project NXT GEN AI METHODS (19A23014S). We thank the International Max Planck Research School for Intelligent Systems (IMPRS-IS) for supporting Bernhard Jaeger, Daniel Dauner, and Kashyap Chitta. We thank Katrin Renz for proofreading.

References

[1] M. Treiber, A. Hennecke, and D. Helbing. Congested traffic states in empirical observations and microscopic simulations. *Physical review E*, 2000.

[2] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, K. Lau, C. M. Oakley, M. Palatucci, V. R. Pratt, P. Stang, S. Strohband, C. Dupont, L. Jendrossek, C. Koelen, C. Markey, C. Rummel, J. van Niekerk, E. Jensen, P. Alessandrini, G. R. Bradski, B. Davies, S. Ettinger, A. Kaehler, A. V. Nefian, and P. Mahoney. Stanley: The robot that won the DARPA grand challenge. *Journal of Field Robotics (JFR)*, 23(9):661–692, 2006.

[3] B. Jaeger. Expert drivers for autonomous driving. Master's thesis, University of Tübingen, 2021.

[4] D. Dauner, M. Hallgarten, A. Geiger, and K. Chitta. Parting with misconceptions about learning-based vehicle motion planning. In *Proc. Conf. on Robot Learning (CoRL)*, 2023.

[5] C. Sima, K. Renz, K. Chitta, L. Chen, H. Zhang, C. Xie, J. Beißwenger, P. Luo, A. Geiger, and H. Li. Drivelm: Driving with graph visual question answering. In *Proc. of the European Conf. on Computer Vision (ECCV)*, 2024.

[6] K. Renz, K. Chitta, O.-B. Mercea, A. S. Koepke, Z. Akata, and A. Geiger. Plant: Explainable planning transformers via object-level representations. In *Proc. Conf. on Robot Learning (CoRL)*, 2022.

[7] M. Hallgarten, M. Stoll, and A. Zell. From prediction to planning with goal conditioned lane graph traversals. In *Proc. IEEE Conf. on Intelligent Transportation Systems (ITSC)*, 2023.

[8] J. Cheng, Y. Chen, X. Mei, B. Yang, B. Li, and M. Liu. Rethinking imitation-based planners for autonomous driving. In *Proc. IEEE International Conf. on Robotics and Automation (ICRA)*, 2024.

[9] Z. Huang, H. Liu, J. Wu, and C. Lv. Differentiable integrated motion prediction and planning with learnable cost function for autonomous driving. *IEEE Trans. Neural Networks Learn. Syst.*, 2024.

[10] Q. Sun, H. Wang, J. Zhan, F. Nie, X. Wen, L. Xu, K. Zhan, P. Jia, X. Lang, and H. Zhao. Generalizing motion planners with mixture of experts for autonomous driving. *arXiv.org*, 2410.15774, 2024.

[11] J. Guo, M. Feng, P. Zhu, C. Li, and J. Pu. Rethinking closed-loop planning framework for imitation-based model integrating prediction and planning. *arXiv.org*, 2407.05376, 2024.

[12] J. Cheng, Y. Chen, and Q. Chen. PLUTO: pushing the limit of imitation learning-based planning for autonomous driving. *arXiv.org*, 2404.14327, 2024.

[13] X. Chen, J. Yan, W. Liao, T. He, and P. Peng. Int2planner: An intention-based multi-modal motion planner for integrated prediction and planning. *arXiv.org*, 2501.12799, 2025.

[14] Z. Zhang, A. Liniger, D. Dai, F. Yu, and L. 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.

[15] C. Zhang, R. Guo, W. Zeng, Y. Xiong, B. Dai, R. Hu, M. Ren, and R. Urtasun. Rethinking closed-loop training for autonomous driving. In *Proc. of the European Conf. on Computer Vision (ECCV)*, 2022.

[16] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[17] B. Jaeger and A. Geiger. An invitation to deep reinforcement learning. *Foundations and Trends in Optimization*, 2024.

[18] W. B. Knox, A. Allievi, H. Banzhaf, F. Schmitt, and P. Stone. Reward (mis)design for autonomous driving. *Artificial Intelligence (AI)*, 2023.

[19] A. Kendall, J. Hawke, D. Janz, P. Mazur, D. Reda, J. Allen, V. Lam, A. Bewley, and A. Shah. Learning to drive in a day. In *Proc. IEEE International Conf. on Robotics and Automation (ICRA)*, 2019.

[20] E. Wijmans, A. Kadian, A. Morcos, S. Lee, I. Essa, D. Parikh, M. Savva, and D. Batra. DD-PPO: learning near-perfect pointgoal navigators from 2.5 billion frames. In *Proc. of the International Conf. on Learning Representations (ICLR)*, 2020.

[21] F. Fuchs, Y. Song, E. Kaufmann, D. Scaramuzza, and P. Dürr. Super-human performance in gran turismo sport using deep reinforcement learning. *IEEE Robotics and Automation Letters (RA-L)*, 2021.

[22] K. Zeng, Z. Zhang, K. Ehsani, R. Hendrix, J. Salvador, A. Herrasti, R. B. Girshick, A. Kembhavi, and L. Weihs. Poliformer: Scaling on-policy RL with transformers results in masterful navigators. In *Proc. Conf. on Robot Learning (CoRL)*, volume 2406.20083, 2024.

[23] P. R. Wurman, S. Barrett, K. Kawamoto, J. MacGlashan, K. Subramanian, T. J. Walsh, R. Capobianco, A. Devlic, F. Eckert, F. Fuchs, L. Gilpin, P. Khandelwal, V. Kompella, H. Lin, P. MacAlpine, D. Oller, T. Seno, C. Sherstan, M. D. Thomure, H. Aghabozorgi, L. Barrett, R. Douglas, D. Whitehead, P. Dürr, P. Stone, M. Spranger, and H. Kitano. Outracing champion gran turismo drivers with deep reinforcement learning. *Nature*, 2022.

[24] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv.org*, 1707.06347, 2017.

[25] M. Toromanoff, E. Wirbel, and F. Moutarde. End-to-end model-free reinforcement learning for urban driving using implicit affordances. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[26] R. Chekroun, M. Toromanoff, S. Hornauer, and F. Moutarde. GRI: general reinforced imitation and its application to vision-based autonomous driving. *Robotics*, 2023.

[27] Q. Li, X. Jia, S. Wang, and J. Yan. Think2drive: Efficient reinforcement learning by thinking with latent world model for autonomous driving (in CARLA-V2). In *Proc. of the European Conf. on Computer Vision (ECCV)*, 2024.

[28] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun. CARLA: An open urban driving simulator. In *Proc. Conf. on Robot Learning (CoRL)*, 2017.

[29] C. team. Carla autonomous driving leaderboard 2.0. https://leaderboard.carla.org/, 2022.

[30] K. Chitta, A. Prakash, B. Jaeger, Z. Yu, K. Renz, and A. Geiger. Transfuser: Imitation with transformer-based sensor fusion for autonomous driving. *Transactions on Pattern Analysis and Machine Intelligence (T-PAMI)*, 2023.

[31] N. Karnchanachari, D. Geromichalos, K. S. Tan, N. Li, C. Eriksen, S. Yaghoubi, N. Mehdipour, G. Bernasconi, W. K. Fong, Y. Guo, and H. Caesar. Towards learning-based planning: The nuplan benchmark for real-world autonomous driving. In *Proc. IEEE International Conf. on Robotics and Automation (ICRA)*, 2024.

[32] Y. Zheng, R. Liang, K. Zheng, J. Zheng, L. Mao, J. Li, W. Gu, R. Ai, S. E. Li, X. Zhan, et al. Diffusion-based planning for autonomous driving with flexible guidance. *Proc. of the International Conf. on Learning Representations (ICLR)*, 2025.

[33] CARLA. Carla autonomous driving leaderboard 2.0 scenarios. https://leaderboard.carla.org/scenarios, 2022.

[34] CARLA. Leaderboard 1.0 to 2.0 scenario converter. https://github.com/carla-simulator/leaderboard/blob/40d507ce67b189e66458a12492acaab706b28e92/scripts/route_bridge.py, 2024.

[35] C. team. Carla leaderboard 2.0 metrics. URL: https://leaderboard.carla.org/evaluation_v2_0, 2024.

[36] A. Prakash, A. Behl, E. Ohn-Bar, K. Chitta, and A. 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.

[37] A. Behl, K. Chitta, A. Prakash, E. Ohn-Bar, and A. Geiger. Label efficient visual abstractions for autonomous driving. In *Proc. IEEE International Conf. on Intelligent Robots and Systems (IROS)*, 2020.

[38] R. Islam, P. Henderson, M. Gomrokchi, and D. Precup. Reproducibility of benchmarked deep reinforcement learning tasks for continuous control. *arXiv.org*, 1708.04133, 2017.

[39] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger. Deep reinforcement learning that matters. In *Proc. of the Conf. on Artificial Intelligence (AAAI)*, 2018.

[40] N. A. Lynnerup, L. Nolling, R. Hasle, and J. Hallam. A survey on reproducibility by evaluating deep reinforcement learning algorithms on real-world robots. In *Proc. Conf. on Robot Learning (CoRL)*, 2019.

[41] R. Agarwal, M. Schwarzer, P. S. Castro, A. C. Courville, and M. G. Bellemare. Deep reinforcement learning at the edge of the statistical precipice. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[42] A. J. Fetterman, E. Kitanidis, J. Albrecht, Z. Polizzi, B. Fogelman, M. Knutins, B. Wróblewski, J. B. Simon, and K. Qiu. Tune as you scale: Hyperparameter optimization for compute efficient training. *arXiv.org*, 2306.08055, 2023.

[43] S. Huang, R. F. J. Dossa, A. Raffin, A. Kanervisto, and W. Wang. The 37 implementation details of proximal policy optimization. *The ICLR Blog Track 2023*, 2023.

[44] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992.

[45] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *Proc. of the International Conf. on Machine learning (ICML)*, 2016.

[46] T. Degris, M. White, and R. S. Sutton. Off-policy actor-critic. *arXiv.org*, 1205.4839, 2012.

[47] D. Zhang, J. Liang, K. Guo, S. Lu, Q. Wang, R. Xiong, Z. Miao, and Y. Wang. Carplanner: Consistent auto-regressive trajectory planning for large-scale reinforcement learning in autonomous driving. *arXiv.org*, 2502.19908, 2025.

[48] V. Koltun. Autonomous driving: The way forward. URL: https://www.youtube.com/watch?v=XmtTjqimW3g, 2020.

[49] W. D. Hillis and G. L. S. Jr. Data parallel algorithms. *Communications of the ACM*, 1986.

[50] E. Kaufmann, L. Bauersfeld, A. Loquercio, M. Müller, V. Koltun, and D. Scaramuzza. Champion-level drone racing using deep reinforcement learning. *Nature*, 2023.

[51] B. Jaeger, K. Chitta, D. Dauner, K. Renz, and A. Geiger. Common mistakes in benchmarking autonomous driving. URL: https://github.com/autonomousvision/carla_garage/blob/leaderboard_2/docs/common_mistakes_in_benchmarking_ad.md, 2024.

[52] D. Hafner, J. Pasukonis, J. Ba, and T. Lillicrap. Mastering diverse control tasks through world models. *Nature*, 2025.

[53] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NIPS)*, pages 5998–6008, 2017.

[54] J. Liu, Z. Yang, Z. Huang, W. Li, S. Dang, and H. Li. Simulation performance evaluation of pure pursuit, stanley, lqr, mpc controller for autonomous vehicles. In *IEEE international conference on real-time computing and robotics (RCAR)*. IEEE, 2021.

[55] T. Miki, J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter. Learning robust perceptive locomotion for quadrupedal robots in the wild. *Science Robotics*, 2022.

[56] T. Lin, K. Sachdev, L. Fan, J. Malik, and Y. Zhu. Sim-to-real reinforcement learning for vision-based dexterous manipulation on humanoids. *arXiv.org*, 2502.20396, 2025.

[57] C. J. Watkins and P. Dayan. Q-learning. *Machine Learning*, 1992.

[58] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. A. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 2015.

[59] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *Proc. of the International Conf. on Machine learning (ICML)*, 2018.

[60] R. Rajamani. *Vehicle dynamics and control*. Springer Science & Business Media, 2011.

[61] P. Polack, F. Altché, B. d'Andréa Novel, and A. de La Fortelle. The kinematic bicycle model: A consistent model for planning feasible trajectories for autonomous vehicles? In *Proc. IEEE Intelligent Vehicles Symposium (IV)*, 2017.

[62] R. Chekroun, T. Gilles, M. Toromanoff, S. Hornauer, and F. Moutarde. MBAPPE: mcts-built-around prediction for planning explicitly. In *Proc. IEEE Intelligent Vehicles Symposium (IV)*, 2024.

[63] B. Yang, H. Su, N. Gkanatsios, T. Ke, A. Jain, J. G. Schneider, and K. Fragkiadaki. Diffusion-es: Gradient-free planning with diffusion for autonomous and instruction-guided driving. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2024.

[64] C. Gulino, J. Fu, W. Luo, G. Tucker, E. Bronstein, Y. Lu, J. Harb, X. Pan, Y. Wang, X. Chen, J. D. Co-Reyes, R. Agarwal, R. Roelofs, Y. Lu, N. Montali, P. Mougin, Z. Yang, B. White, A. Faust, R. McAllister, D. Anguelov, and B. Sapp. Waymax: An accelerated, data-driven simulator for large-scale autonomous driving research. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

[65] P. Sun, H. Kretzschmar, X. Dotiwalla, A. Chouard, V. Patnaik, P. Tsui, J. Guo, Y. Zhou, Y. Chai, B. Caine, V. Vasudevan, W. Han, J. Ngiam, H. Zhao, A. Timofeev, S. Ettinger, M. Krivokon, A. Gao, A. Joshi, Y. Zhang, J. Shlens, Z. Chen, and D. Anguelov. Scalability in perception for autonomous driving: Waymo open dataset. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[66] L. Xiao, J. Liu, X. Ye, W. Yang, and J. Wang. Easychauffeur: A baseline advancing simplicity and efficiency on waymax. *arXiv.org*, 2408.16375, 2024.

[67] V. Charraut, T. Tournaire, W. Doulazmi, and T. Buhet. V-max: Making rl practical for autonomous driving. *arXiv.org*, 2503.08388, 2025.

[68] M. Cusumano-Towner, D. Hafner, A. Hertzberg, B. Huval, A. Petrenko, E. Vinitsky, E. Wijmans, T. Killian, S. Bowers, O. Sener, P. Krähenbühl, and V. Koltun. Robust autonomy emerges from self-play. *arXiv.org*, 2502.03349, 2025.

[69] S. Ross, G. J. Gordon, and D. Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Conference on Artificial Intelligence and Statistics (AISTATS)*, 2011.

[70] L. Chen, P. Wu, K. Chitta, B. Jaeger, A. Geiger, and H. Li. End-to-end autonomous driving: Challenges and frontiers. *Transactions on Pattern Analysis and Machine Intelligence (T-PAMI)*, 2024.

[71] Y. Lu, J. Fu, G. Tucker, X. Pan, E. Bronstein, R. Roelofs, B. Sapp, B. White, A. Faust, S. Whiteson, D. Anguelov, and S. Levine. Imitation is not enough: Robustifying imitation with reinforcement learning for challenging driving scenarios. In *Proc. IEEE International Conf. on Intelligent Robots and Systems (IROS)*, 2023.

[72] X. Liang, T. Wang, L. Yang, and E. P. Xing. CIRL: controllable imitative reinforcement learning for vision-based self-driving. In *Proc. of the European Conf. on Computer Vision (ECCV)*, 2018.

[73] Y. Song, H. Lin, E. Kaufmann, P. Dürr, and D. Scaramuzza. Autonomous overtaking in gran turismo sport using curriculum reinforcement learning. In *Proc. IEEE International Conf. on Robotics and Automation (ICRA)*, 2021.

[74] Z. Zhang, A. Liniger, D. Dai, F. Yu, and L. V. Gool. End-to-end urban driving by imitating a reinforcement learning coach. *arXiv.org*, 2108.08265v3, 2021.

[75] P. Wu, X. Jia, L. Chen, J. Yan, H. Li, and Y. Qiao. Trajectory-guided control prediction for end-to-end autonomous driving: A simple yet strong baseline. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

[76] X. Jia, P. Wu, L. Chen, J. Xie, C. He, J. Yan, and H. 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.

[77] X. Jia, Y. Gao, L. Chen, J. Yan, P. L. Liu, and H. Li. Driveadapter: Breaking the coupling barrier of perception and planning in end-to-end autonomous driving. In *Proc. of the IEEE International Conf. on Computer Vision (ICCV)*, 2023.

[78] S. Ishida, G. Corrado, G. Fedoseev, H. Yeo, L. Russell, J. Shotton, J. F. Henriques, and A. Hu. Langprop: A code optimization framework using large language models applied to driving. In *ICLR 2024 Workshop on Large Language Model (LLM) Agents*, 2024.

[79] S. Huang, R. F. J. Dossa, C. Ye, J. Braga, D. Chakraborty, K. Mehta, and J. G. M. Araújo. Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms. *Journal of Machine Learning Research (JMLR)*, 2022.

[80] D. Ha and J. Schmidhuber. Recurrent world models facilitate policy evolution. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[81] D. Hafner, T. P. Lillicrap, J. Ba, and M. Norouzi. Dream to control: Learning behaviors by latent imagination. In *Proc. of the International Conf. on Learning Representations (ICLR)*, 2020.

[82] D. Hafner, T. P. Lillicrap, M. Norouzi, and J. Ba. Mastering atari with discrete world models. In *Proc. of the International Conf. on Learning Representations (ICLR)*, 2021.

[83] S. Kazemkhani, A. Pandya, D. Cornelisse, B. Shacklett, and E. Vinitsky. Gpudrive: Data-driven, multi-agent driving simulation at 1 million FPS. In *Proc. of the International Conf. on Learning Representations (ICLR)*, 2025.

[84] D. Chen and P. Krähenbühl. Learning from all vehicles. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[85] J. Weng, M. Lin, S. Huang, B. Liu, D. Makoviichuk, V. Makoviychuk, Z. Liu, Y. Song, T. Luo, Y. Jiang, Z. Xu, and S. Yan. Envpool: A highly parallel reinforcement learning environment execution engine. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

[86] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, J. Oh, D. Horgan, M. Kroiss, I. Danihelka, A. Huang, L. Sifre, T. Cai, J. P. Agapiou, M. Jaderberg, A. S. Vezhnevets, R. Leblond, T. Pohlen, V. Dalibard, D. Budden, Y. Sulsky, J. Molloy, T. L. Paine, Ç. Gülçehre, Z. Wang, T. Pfaff, Y. Wu, R. Ring, D. Yogatama, D. Wünsch, K. McKinney, O. Smith, T. Schaul, T. P. Lillicrap, K. Kavukcuoglu, D. Hassabis, C. Apps, and D. Silver. Grandmaster level in starcraft II using multi-agent reinforcement learning. *Nature*, 2019.

[87] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Debiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, R. Józefowicz, S. Gray, C. Olsson, J. Pachocki, M. Petrov, H. P. de Oliveira Pinto, J. Raiman, T. Salimans, J. Schlatter, J. Schneider, S. Sidor, I. Sutskever, J. Tang, F. Wolski, and S. Zhang. Dota 2 with large scale deep reinforcement learning. *arXiv.org*, 1912.06680, 2019.

[88] A. Petrenko, Z. Huang, T. Kumar, G. S. Sukhatme, and V. Koltun. Sample factory: Egocentric 3d control from pixels at 100000 FPS with asynchronous reinforcement learning. In *Proc. of the International Conf. on Machine learning (ICML)*, 2020.

[89] Python. Pep 703 – making the global interpreter lock optional in cpython. URL: https://peps.python.org/pep-0703/, 2025.

[90] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Z. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[91] C. Lyle, Z. Zheng, E. Nikishin, B. Á. Pires, R. Pascanu, and W. Dabney. Understanding plasticity in neural networks. In *Proc. of the International Conf. on Machine learning (ICML)*, Proceedings of Machine Learning Research, 2023.

[92] S. Dohare, J. F. Hernandez-Garcia, Q. Lan, P. Rahman, A. R. Mahmood, and R. S. Sutton. Loss of plasticity in deep continual learning. *Nature*, 2024.

[93] A. Juliani and J. Ash. A study of plasticity loss in on-policy deep reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

[94] S. Moalla, A. Miele, D. Pyatko, R. Pascanu, and C. Gulcehre. No representation, no trust: Connecting representation, collapse, and trust issues in PPO. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

[95] P. Chou, D. Maturana, and S. A. Scherer. Improving stochastic policy gradients in continuous control with deep reinforcement learning using the beta distribution. In *Proc. of the International Conf. on Machine learning (ICML)*, 2017.

[96] I. G. B. Petrazzini and E. A. Antonelo. Proximal policy optimization with continuous bounded action space via the beta distribution. In *IEEE Symposium Series on Computational Intelligence, (SSCI)*, 2021.

[97] B. Jaeger, K. Chitta, and A. Geiger. Hidden biases of end-to-end driving models. In *Proc. of the IEEE International Conf. on Computer Vision (ICCV)*, 2023.

[98] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym. *arXiv.org*, 1606.01540, 2016.

[99] M. Towers, A. Kwiatkowski, J. K. Terry, J. U. Balis, G. D. Cola, T. Deleu, M. Goulão, A. Kallinteris, M. Krimmel, A. KG, R. Perez-Vicente, A. Pierré, S. Schulhoff, J. J. Tai, H. Tan, and O. G. Younis. Gymnasium: A standard interface for reinforcement learning environments. *arXiv.org*, 2407.17032, 2024.

[100] P. Hintjens. *ZeroMQ: Messaging for Many Applications*. O'Reilly Media, 2013.

[101] S. Huang, R. F. J. Dossa, A. Raffin, A. Kanervisto, and W. Wang. The 37 implementation details of proximal policy optimization. In *ICLR Blog Track*, 2022. URL https://iclr-blog-track.github.io/2022/03/25/ppo-implementation-details/.

[102] N. Hanselmann, K. Renz, K. Chitta, A. Bhattacharyya, and A. Geiger. King: Generating safety-critical driving scenarios for robust imitation via kinematics gradients. In *Proc. of the European Conf. on Computer Vision (ECCV)*, 2022.

[103] K. Hao, W. Cui, Y. Luo, L. Xie, Y. Bai, J. Yang, S. Yan, Y. Pan, and Z. Yang. Adversarial safety-critical scenario generation using naturalistic human driving priors. *IEEE Trans. on Intelligent Transportation Systems (TITS)*, 2023.

[104] Y. Yin, P. Khayatan, Éloi Zablocki, A. Boulch, , and M. Cord. Regents: Real-world safety-critical driving scenario generation made stable. In *ECCV 2024 W-CODA Workshop*, 2024.

[105] D. R. Hipp. Sqlite. URL: http://sqlite.org/, 2000.

[106] J. V. den Bossche, K. Jordahl, M. Fleischmann, M. Richards, J. McBride, J. Wasserman, A. G. Badaracco, A. D. Snow, B. Ward, J. Tratner, J. Gerard, M. Perry, cjqf, G. A. Hjelle, M. Taves, E. ter Hoeven, M. Cochran, R. Bell, rraymondgh, M. Bartos, P. Roggemans, L. Culbertson, G. Caria, N. Y. Tan, N. Eubank, sangarshanan, J. Flavin, S. Rey, and J. Gardiner. geopandas/geopandas: v1.0.1, 2024. URL https://doi.org/10.5281/zenodo.12625316.

[107] J. Zimmerlin, J. Beißwenger, B. Jaeger, A. Geiger, and K. Chitta. Hidden biases of end-to-end driving datasets. *arXiv.org*, 2412.09602, 2024.

[108] D. Hafner, J. Pasukonis, J. Ba, and T. Lillicrap. Mastering diverse domains through world models. *arXiv.org*, 2023.

[109] O. Scheel, L. Bergamini, M. Wolczyk, B. Osiński, and P. Ondruska. Urban driver: Learning to drive from real-world demonstrations using policy gradients. In *Proc. Conf. on Robot Learning (CoRL)*, 2021.

CaRL: Learning Scalable Planning Policies with Simple Rewards

Supplementary Material

A Related work

Driving simulations: We use simulations to train and evaluate our methods in this work. **CARLA** [28] is a 3D open source autonomous driving simulator, based on the Unreal Engine, enabling real-time graphical rendering and simulation. CARLA enables research on the full driving stack from perception to planning and control, and has an extensive set of functionalities extended and developed over the last 8 years. The environments, assets, and scenarios are designed to offer a high-definition simulation of safety-critical events with arbitrary duration. In return, CARLA is considered computationally expensive, due to the 3D simulation and flexibility of the simulator. In this work, we use the simulation code of the CARLA leaderboard 2.0 together with the longest6 v2 benchmark [30].

nuPlan [31] is a data-driven simulator for vehicle motion planning that uses the real-world nuPlan dataset for simulation initialization or replay-based background traffic. As such, nuPlan simulates short scenes (i.e.15s) in a lightweight 2D setting on real-world maps with a kinematic bicycle model [60, 61]. Compared to CARLA, nuPlan uses fewer resources at the cost of complexity. We use the CARLA leaderboard 2.0 to benchmark long-form driving with safety-critical scenarios and complement the results with nuPlan to test realistic everyday driving situations. A particularly interesting result of the nuPlan simulator and its benchmark was that it showed that rule-based [4] and search-based [62, 63] methods still outperform pure learning-based approaches [7, 8, 12].

Waymax [64] is another recent bird's-eye-view (BEV) data-driven simulator, like nuPlan. It is different in that it uses a different underlying dataset [65] and is implemented in Jax, which allows for GPU acceleration. It otherwise has the same strengths and weaknesses as nuPlan. Several concurrent works [66, 67, 68] have reported that route information needed for proper simulation is currently proprietary in the WaymoOpen dataset. As a workaround, these works have used the human expert trajectory as conditioning input to the model. This can leak label information about an optimal trajectory to the policy, potentially leading to misleading results. Examples include leaking the lane the human expert drove on [68], or even a fine-grained human path [67], trivializing lateral planning. Due to this problem and because Waymax doesn't seem to offer fundamental advantages compared to nuPlan for benchmarking purposes, we are not using this benchmark in this work. In nuPlan, conditioning is based on road blocks, which masks information about precise locations or which lane the human expert drove on.

Reinforcement Learning: Deep Reinforcement Learning [16, 17] (RL) methods train neural networks, referred to as policy, model, or agent, by learning from trial and error. The goal and hence optimization objective of the trial and error learning is to maximize the (discounted) sum of rewards. The RL framework offers two unique advantages compared to popular behavior cloning for the problem of driving. First, RL methods learn online, which avoids the compounding error problem [69] typical for behavior cloning methods. Second, RL methods can optimize non-differentiable rewards [17], which makes it possible to align the optimization objective closely with the desired target metric.

Rewards for Driving: Due to the flexibility in optimization, there are many ways to define the reward for driving, making reward design an important open problem [18, 70].

Early work on RL methods in autonomous driving [19] focused on simplified settings such as lane keeping on an empty street. This simplified setting allowed the methods to use simple principled reward functions such as maximizing forward speed and ending the episode upon an infraction.

Similarly, such simple reward functions are also popular in other robotics fields, such as autonomous racing [21] or indoor navigation [20, 22], but only in settings where there are no other dynamic actors in the environment.

Lu et al. [71] optimize a simple reward, with only collision and an off-road penalty, in dynamic traffic. This reward does not include any term for progress, so a policy that always stays stationary would be optimal in reactive traffic. The model only learns to drive because it is additionally trained with behavior cloning and because other cars will crash into a static ego car in log replay traffic. Therefore, such a reward is not reasonable for pure RL in reactive traffic, which is the problem we investigate in this work.

When training with RL, in settings with dynamic actors, such as other cars, it is reported that simple rewards do not suffice anymore [72, 25, 73, 23]. To address this, complex reward terms are engineered that give additional local signals to the learning algorithm. This makes optimization easier, because it simplifies the credit assignment problem, allowing weaker learning recipes to succeed but potentially upper bounds performance as the reward may not be well aligned anymore with the original objective.

Toromanoff et al. 2020 [25] was the first RL method to train a planning policy head from scratch with RL in the dynamic and reactive traffic of the CARLA leaderboard 1.0. They proposed to reward the difference between a desired vehicle state and the current vehicle state (desired position, rotation, and speed). This desired vehicle state is computed with privileged rules and provides very dense information for the learning agent. This provides dense feedback, simplifying the learning problem, but limits the agent's performance to the performance of the rule-based planner used to compute the reward. Due to its success, this general reward design became the dominant paradigm for state-of-the-art RL methods in CARLA [14, 26, 27]. Our work shows that this trade-off is not necessary. We propose a more principled reward, free of rule-based experts, and show that it enabled data scaling via large mini-batch sizes.

RL Planners: **Roach** [14] is the first method to train a planner in CARLA with PPO [24] uses a similar reward to [25], which computes its reward based on the difference to a desired vehicle state. While the paper worked with the CARLA simulator, it is important to note that the conference paper erroneously claims SOTA performance on the CARLA leaderboard. The method was instead trained and evaluated on a newly created CARLA benchmark, which reused the route definitions of the CARLA leaderboard but did not include any safety-critical scenarios. This was later corrected in version 3 of the ArXiv [74] report. A major problem with Roach was that its open-source training code is incompatible with the CARLA scenario runner that is used to evaluate safety-critical scenarios. Some works made the inference code compatible [75, 76, 77], but the model achieved low scores when evaluated with safety-critical scenarios [78] because it has never encountered these scenarios during training. Methods that used Roach for data collection combined the model with rule-based planners to post-process its actions. Instead of this workaround, we rewrote the training code from scratch for the CARLA leaderboard 2.0. We build our code upon CleanRL [79] and use the open-source Roach codebase to match implementation details precisely. We reproduce the Roach method on the CARLA leaderboard 2.0 longest6 v2 benchmark. We observe that Roach achieves much lower scores than rule-based [5] or imitation learning [6] based baselines. We build upon Roach and show that PPO can optimize our more principled reward when the mini-batch size is increased.

Think2Drive [27] is a recent RL planner trained with the CARLA leaderboard 2.0 that advocates for the use of world models [80, 81, 82, 52] instead of using PPO. World models simulate the original simulator with a faster neural network-based simulator. This is necessary because the default CARLA leaderboard 2.0 is too slow to converge PPO. The resulting world model simulator is imperfect but able to produce more samples in the same wall clock time. The policy is then trained exclusively with the approximated data from the world model. Using World models in a simulator is a compromise that trades off larger sample efficiency against lower asymptotic performance of the policy, because flaws in the world model translate to flaws in the policy [17]. We show that

this compromise is not necessary on CARLA. We demonstrate that a two orders of magnitude increase in training throughput can be achieved through a combination of software engineering, better hyperparameters, scale, and a more efficient scaling technique in the CARLA leaderboard 2.0.

Think2Drive claims to achieve 56.8 Driving Score on the official test routes of the CARLA leaderboard 2.0 test routes. It is not described what these "official" test routes refer to. The CARLA leaderboard 2.0 has an official test server, however, it can only be used for sensorimotor agents. Privileged agents like Think2Drive cannot be evaluated on it, because the terms of service explicitly prohibit it. The CARLA leaderboard 2.0 has a set of 2 routes called DevTest, which are in the training town 12 and are likely meant for development testing (debugging). Evaluating on these or the more principled validation routes (which contain 20 routes and are on the validation town 13) requires solving roughly 90 safety-critical scenarios consecutively along roughly 10 km-long routes. Think2Drive reports an average success rate per scenario of 84% (average of Table 2 in the paper). One can estimate the Driving Score when solving 90 scenarios consecutively at an 84% individual success rate. We ran a simple Monte Carlo simulation, averaging 10000 trials assuming perfect route completion and no other infractions, and using the median penalty factor of 0.65. This resulted in an average of 14 infractions per route and an average of 0.6 DS. This is much lower than the reported 56.8 DS, which makes it unlikely that the unmodified devtest or validation routes are meant. The authors uploaded a video of the model driving on the test routes. The video does not showcase that the model can solve multiple scenarios consecutively, but instead cuts after every individual scenario.

The code and model of Think2Drive are not public, so it is not directly possible to verify the claims made in the paper. To alleviate this issue, we reproduce Think2Drive on the CARLA leaderboard 2.0 longest 6 v2 benchmark and will publish the code and models.

GPUDrive [83] and **GIGAFLOW** [68] are concurrent works that speed up RL for planning by coding GPU-accelerated driving simulators. These driving simulators approximate other simulators by removing functionality and approximating expensive computations. GPUDrive approximates Waymax [64] using the assets of the Waymo Open dataset [65], whereas GIGAFLOW uses the map assets from CARLA [28]. In some sense, writing approximate faster simulators is similar to learning world models, with the advantage that with manual coding, the approximation errors can be controlled precisely. Since the approximate simulator might be easier to solve than the original problem, it is important to evaluate on the original benchmark after training with the approximate simulator. In this work, we similarly train with a faster version of the CARLA leaderboard 2.0, but evaluate with the original code. GPUDrive does not evaluate its policy on Waymax and has no non-RL baselines, so it is hard to evaluate the performance of the policy. GIGAFLOW, on the other hand, demonstrates Sim2Sim transfer evaluating their policy on nuPlan, Waymax, and the CARLA leaderboard 1.0. While the GIGAFLOW training speed is remarkably efficient, achieving over 1 million FPS training throughput, their learning recipe is remarkably inefficient, requiring 1 trillion samples to converge. Our best model (see Section E.2) achieves similar scores (93.1 vs 93.8) on nuPlan val14 reactive with 1 billion samples, 1000 times less data, while using the same learning algorithm [20]. Since GIGAFLOW simulates faster, this leads to similar overall compute requirements. Our method is perhaps a better candidate for future extensions to end-to-end RL with vision because our 1B samples requirement can be achieved with a roughly 1000 FPS training throughput. Some works have achieved such throughputs with vision in indoor navigation simulators [20, 22]. Training the deep networks required for vision at the 1 Million FPS requirement of GIGAFLOW might need LLM pre-training level compute.

GIGAFLOW tries to address the problem of trading off different reward components by aggressively randomizing the tradeoff parameters. They learn a variety of policies by conditioning the model on the reward parameters. The parameters then need to be tuned at inference to recover a good planner. This leads to a strong driving policy, but might increase sample requirements as many policies need to be learned. Instead, we propose to eliminate the reward tradeoffs by optimizing a single reward component.

Both GIGAFLOW and GPUDrive learn from all vehicles [84] because the policy is trained via self-play to control all actors in the scene. The number of samples used in both works is multiplied by

the number of actors in each simulator because of that. Whether multi-agent samples have the same value as time step samples is unclear, since they could also be viewed as augmented versions of the same sample. Self-play is a complementary technique to our work that can make a policy robust to varying traffic behaviors and therefore a promising approach for future attempts at Sim2Real transfer.

Asynchronous collection: PPO synchronizes many parallel environments by running batch model forward passes during data collection, which can lead to idle resources if the time of environment steps is variable. Envpool [85] proposed to alleviate the PPO collection synchronization problem by introducing a collection mini-batchsize where only a minibatch of ready environments is forward passed at a time. This alleviates the synchronization problem by reducing the number of synchronized environments, but has the downside that the number of sequential forward passes needed to collect the data is increased, which can slow down data collection in cases where the environment step times are similar. Additionally, it oversamples data from fast environments, which in CARLA are situations with fewer vehicles, hence easier samples. Our asynchronous collection solution, proposed in Section B.1, does not bias the data collection towards faster environments and keeps the number of sequential forward passes to the network the same.

Various works have proposed fully asynchronous policy gradient methods [45, 86, 87, 88], which are methods that do not synchronize the backward pass. These methods train faster but often do not study the effect of policy lag. By policy lag, we mean the *variable and uncontrollable* amount of off-policy policy gradient error that gets introduced if training and data collection are decoupled. This makes it unclear whether fully asynchronous methods outperform approaches that do synchronize the backward pass, given a fixed compute budget. Anecdotal evidence [87] (figure 5b, policy lag is called data staleness) suggests that the negative effect of policy lag can be quite severe. A more rigorous comparison between backward synchronized and fully asynchronous policy gradient methods is an interesting direction for future work. Our proposed method, AC-PPO, synchronizes the backward pass during training which avoids the problem of policy lag.

B Method changes

For our final model, trained with 300M samples, we have made several changes to Roach. In particular, increasing the field of view (FOV) of the input is important to achieve Driving Scores > 50 on longest6 v2, although it decreases performance in low data regimes, since the task becomes harder with the resulting smaller objects and larger amount of pixels. The following tables are structured such that each row is the baseline for the next. We found these changes helpful in preliminary experiments, although some of them have little impact on the model trained with a 1024 mini-batch size since it seems to already be close to the best performance that it can get given its limited FOV.

B.1 Asynchronous data collection with PPO

PPO parallelizes data collection across multiple environments. In environments whose time steps take a variable amount of time, also called nonhomogeneous environments [20], PPO is suboptimal in terms of efficiency [85]. PPO performs batched forward passes with the model during data collection, which synchronizes each environment after each environment step. CARLA is a nonhomogeneous environment, where the speed of a step varies between towns and traffic densities. Additionally, environment resets are significantly slower. This means all parallel environments wait for the slowest, leading to unnecessary idle times.

We propose to remove the collection synchronization, running each forward pass concurrently with batch size 1 in parallel. We name this approach Asynchronous Collection Proximal Policy Optimization (AC-PPO). Running multiple model forward passes concurrently on a GPU efficiently requires using multi-threading and CUDA streams to share the model weights and CUDA contexts. Unfortunately, Python does not support concurrent multi-threading because the global interpreter lock prevents concurrent execution of threads. This is a known limitation of the Python programming

Method	FPS ↑	Train time h ↓
PPO [24]	130	22
AC-PPO	**144**	**19**

Table 7: **PPO vs AC-PPO**

language, and the developers are actively working on improving it [89]. For now, AC-PPO can be implemented in other languages. We re-implemented our training code in C++, using the LibTorch interface of PyTorch [90]. Table 7 shows that AC-PPO trains 11 percent faster than PPO using the same hardware and performing the same computations (up to random seeds). This reduces training time by 3 hours.

B.2 Hyperparameter

We further reduced the number of training epochs from 4 (Atari) to 3 in CARLA. As Table 8 shows, this increased DS by 2 points and slightly reduced training time.

Epochs	DS ↑	RC ↑
4	38 ± 3	**85** ± 5
3	**40** ± 2	84 ± 1

Table 8: **Effect of reducing training epochs.**

The full list of hyperparameters used in Atari, Roach, and our method is displayed in Table 9.

Hyperparameter	**Roach** [14]	**Atari** [24, 43]	**CaRL**(CARLA)	**CaRL**(nuPlan)
Samples	10M	10M	300M	500M/**1B**
Initial Learning rate	0.00001	**0.00025**	**0.00025**	**0.00025**
Learning rate schedule	KL patience	linear	linear	linear
num envs	6	8	128	512/**1024**
env steps per iteration	**2048**	128	512	32/32
batch size	12288	1024	**65536**	16384/32768
mini-batch size	256	256	**16384**	4096/8192
Steps per epoch	**48**	4	4	4
Epochs	**20**	4	3	4
Steps off-policy	**959**	15	11	15
Exploration Hints	✓	✗	✗	✗
norm advantage	✗	✓	✓	✓
clip value loss	✗	✓	✓	✓
entropy coefficient	**0.01**	**0.01**	**0.01**	**0.01**
value function coefficient	**0.5**	**0.5**	**0.5**	**0.5**
discount factor	**0.99**	**0.99**	**0.99**	**0.99**
GAE λ	0.9	**0.95**	**0.95**	**0.95**
clipping coefficient	**0.2**	0.1	0.1	0.1
Max gradient norm	**0.5**	**0.5**	**0.5**	**0.5**

Table 9: **PPO hyperparameters used in Atari, Roach and CaRL**

B.3 Architecture

We follow the architecture of [14], consisting of 6 convolutional layers for the bird's-eye-view (BEV) semantic image, encode the measurements with 2 layer MLPs, merge the features with a 2 layer MLP and decode the features with a 2 layer MLP into α and β values for the action probablity distribution and a seperate MLP for the value head. All layers are followed by a ReLU activation function.

In addition to this basic architecture, we found it helpful to introduce a layernorm before every ReLU activation function.

Architecture	DS ↑	RC ↑
Cnn [14]	40 ± 2	84 ± 1
+ Layernorm	**42** ± 4	93 ± 2
+ Asymmetric Critic	41 ± 3	**94** ± 3

Table 10: **Architecture.**

Table 10 shows that layernorm improved the driving score by 2 points and the route completion by 9 points. One hypothesis for why the layernorm is helpful might be that it increases the plasticity of the network [91]. Maintaining plasticity is particularly important in continual learning tasks [92, 93]. On-policy reinforcement learning is a non-stationary optimization problem, because if the policy changes, so will the states it observes. To fit the new states, the network needs to maintain plasticity [94]. By increasing plasticity layernorm might be a generally helpful tool for on-policy RL.

Additionally, Table 10 shows the impact of using an asymmetric critic. The result is similar, with the driving score reduced by 1 point and the route completion increased by 1 point. Asymmetric critic means that the value function gets additional information that the policy doesn't know. These are inputs that are unnecessary for driving but help predict the cumulative rewards. In particular, we use the time until a timeout happens, a blocked infraction happens, the remaining length of the route, how long any time to collision penalty is still applied, and for how many frames a comfort penalty is still applied, all appropriately normalized.

B.4 Action distribution

The action space of the car is steering $\in [-1, 1]$, throttle $\in [0, 1]$ and brake $\in [0, 1]$. Throttle and brake are merged into one action dimension $\in [-1, 1]$, making them mutually exclusive. These continuous values require discretization or a continuous probability distribution. Zhang et al. [14] report that the Normal distribution performs much worse than the Beta distribution, which our preliminary experiments confirmed.

The Beta distribution introduces an additional choice: The Beta distribution is parametrized by two parameters $\alpha, \beta \in (0, \infty)$. For $\alpha, \beta \in (0, 1)$, the distribution assigns a lot of probability to the extreme ends of the bounded action space and can become multi-modal. This overweights extreme actions such as full throttle or brake, which should be used very rarely as they lead to uncomfortable driving. Additionally, the distribution degenerates as $\alpha, \beta \to 0$. Instead of using the full range of α, β like [14], we restrict the network to predict values $\in [1, \infty)$ by adding 1 to the Softplus activation function, following [95, 96]. This ensures that the distribution is unimodal and does not degenerate as $\alpha, \beta \to \infty$ is needed for the distribution to collapse.

Action distribution	DS ↑	RC ↑
Beta $\alpha, \beta \in (0, \infty)$	**41** ± 3	**94** ± 3
Beta $\alpha, \beta \in (1, \infty)$	**41** ± 2	**94** ± 2

Table 11: **Action distribution parameterization**

Table 11 shows that both variants achieve the same driving score. The driving score does not consider comfort, so improvements in comfort are not measured.

Fig. 2 shows 4 different shapes of the Beta distribution PDF to give the reader some intuition. The α, β values are displayed in the title of the plot. The first and third plots show examples for $\alpha, \beta <= 1$. The distribution can become multimodal (first plot) or degenerate towards either extreme

(third plot). When $\alpha, \beta >= 1$, then the distribution behaves more like a normal distribution, if $\alpha \approx \beta$ (second plot), and does not degenerate quickly if $\alpha < \beta$ or vice versa (fourth plot).

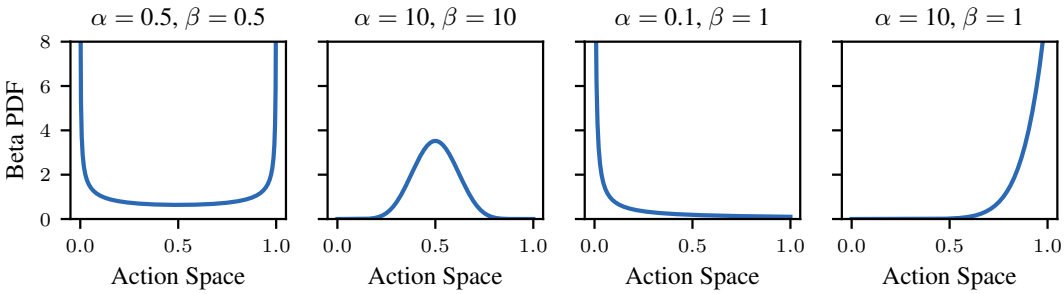


Figure 2: **Beta distribution examples.**

B.5 Input

Similar to prior work [14, 27], we use a BEV semantic segmentation image as input. The channels consist of a channel for the road, an A* route mask, lane markings, vehicles, pedestrians, traffic lights, stop signs, speed signs, static objects, and shoulder lanes. Unlike prior work [14, 27], we do not use additional channels for the states of pedestrians and vehicles from past time steps. Instead of using a binary mask for the vehicles and pedestrians, we encode their speed value in the bounding box. This reduces GPU memory requirements. Similar to concurrent work [68], we find historical information, aside from the speed of vehicles, to not be necessary for the CARLA and nuPlan benchmarks. Additionally, we render open doors, a constant velocity forecast, the blinker, stop, and emergency lights of the car into the car channel via special bounding boxes.

The route mask channel renders the lane-level route from the CARLA A* route planner that determines where the car should go. This is meant to condition the model such that it knows where it should drive at intersections, similar to the target points used in sensor-based agents [97], but much denser. We observed that even if the reward function allows the agent to deviate from the lanes that the A* planner picked, it would still follow the A* lane most of the time. This is likely because the model learns to ignore the lane markings and road channel and mostly abuses the A* route mask channel for steering. This leads to the same pathologies that the other CARLA baselines have (see Section C), where the agent lane changes at pre-defined locations, leading to crashes on highways, even if the reward does not directly enforce the behavior as done in prior work. These models have the characteristic 0.0 route deviation infractions, which are also typical for imitation learning models that abuse a similar bias [97]. We were able to break this behavior by only rendering the A* route inside intersections, which enabled the policy to learn how to choose its driving lane by itself even inside intersections where there is conditioning. This reduced the mentioned lane change collision, however, it also had the downside that the agent now sometimes takes the wrong turn at intersections.

The current policy incurs many infractions in situations where the FOV of the input is too small to see fast approaching cars in time. Roach used 30m front, 8 m to the back, and 19m to each side. This might have been fine for low-speed driving, but when driving at higher speeds, the car needs to be able to see other vehicles earlier since the time to stop the vehicle increases roughly quadratically with driving speed. We increased the visible range to 78 m in front, 50 m in back, and 64 m to either side. This would naively make the input image much larger. To stay in a similar range, we decreased the pixels per meter resolution from 5 to 2 and increased image resolution from 192x192 to 256x256. To account for the increase in resolution, we added an additional Conv2D layer with stride 2 to the network to keep the number of output features from the CNN constant. Our final model has 2 million parameters.

Similar to prior work we also use scalar measurements as input to the policy. We use the last steering, throttle, and brake of the car, the gear state, velocity, speed as well as the current speed limit. As

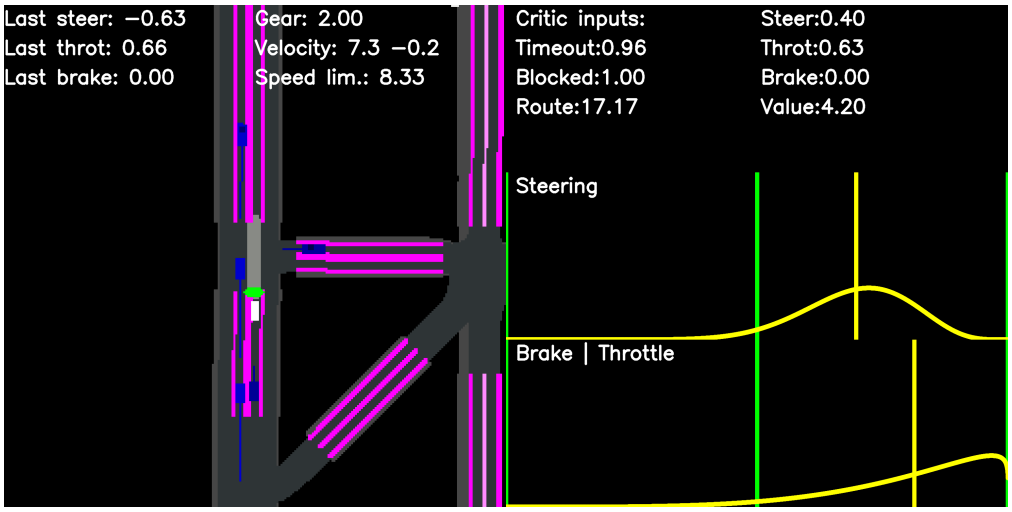


Figure 3: **A rendering of our input**. The distribution shows the model action distribution predictions. The yellow vertical line denotes the mean of the distribution. Other cars are rendered in blue. The brightness of blue encodes their speed. A constant velocity forecast is rendered as a line in front of other vehicles. The ego car is depicted in white. Conditioning is in light grey and only rendered inside intersections. Dark grey depicts the road. The lane markings are pink.

mentioned before, we additionally use extra scalar measurements that are input only to the value function, which may be helpful for value estimation but are unnecessary for driving. Fig. 3 shows an example of a rendering of the BEV observation.

Table 12 shows that using the observation with the larger field of view and no conditioning outside junctions decreases the driving score by 10 points and increases variance. A closer look at the auxiliary metrics reveals that our input reduces vehicle collisions (Veh) by 0.5, because the policy can choose where to lane change and sees other vehicles earlier. The driving score is worse because of a large drop of 31 points in RC which the driving score puts a strong emphasis on. Without the conditioning bias, the model is not consistent at taking intersections, leading to an increase of 0.3 in route deviations (Dev). Our observation is preferable, since, as shown in the main paper, a lot of the lost route completion will be recovered by scaling up samples and mini-batch size.

BEV Style	Size HWC	Front	Back	L/R	DS ↑	RC ↑	Veh ↓	Dev ↓
Roach [14]	192x192x15	30m	8m	19m	**41** ± 2	**94** ± 2	1.41	**0.00**
Ours	256x256x10	**78m**	**50m**	**64m**	31 ± 7	63 ± 8	**0.91**	0.30

Table 12: **Input.** Our input has a larger Field of view but at lower resolution (2 pixels per meter instead of 5). This makes driving harder but enables the model to avoid collisions that are otherwise impossible to solve.

B.6 nuPlan

This section provides the implementation details for training CaRL on nuPlan.

Hyperparamters: Compared to CARLA, a single nuPlan environment has a low memory and compute footprint, with fewer crashes during training. This enabled us to scale the number of parallel training environments for faster data collection and train with 500M/1B samples. We use the regular number of 4 training epochs. All parameters can be found in Table 9.

Architecture: We use the same architectecture as described in Section B.3 and merely adapt the input layer dimensions.

Action Distribution: The kinematic bicycle model of nuPlan takes the longitudinal acceleration (in $\mathrm{m/s^2}$) and steering rate (in $\mathrm{rad/s}$) of the planner output to propagate the ego agent. Most methods using the nuPlan simulator predict waypoints as an output representation because this was a requirement of the 2023 nuPlan challenge [31]. Limiting the design space for output representations of planners to waypoint trajectories may limit their performance because the representation, while recently successful, is both arbitrary and ambiguous [97]. Since the input to the vehicle model is not waypoints, these predictions are converted by processing the trajectory with an LQR controller. We demonstrate that this choice of trajectory representation, combined with LQR controller, can reduce the performance of the planner by comparing the performance of the human ground truth trajectory with three different controllers in Section E.2.

Instead of using a modular trajectory + controller solution, we integrate the control into the planner in this work, such that the neural network also has to learn control. We map the action distribution from Section B.4 to values for the bicycle model. We scale action output from $[-1, 1]$ to $[-3.2, 2.4]$ $\mathrm{m/s^2}$ and $[-0.84, 0.84]$ rad, for the target acceleration and steering respectively. These limits were derived by applying the default controller on a large set of human trajectories. Next, we use a discrete-time derivative to convert the target steering angle to a steering rate and clip the longitudinal acceleration to restrict reverse driving.

Input: We conceptually use the same input and extent as in CARLA from Section B.5, with minor adjustments for nuPlan. For the route, we render the polygons of the route roadblocks that are provided as route information to a planner in nuPlan. We render speed limits and traffic-light states as polylines of the lane center. Since nuPlan does not consider stop-sign infractions, we dropped the corresponding channel to reduce rendering time. We adapt the scalar measurements and input the last steering and acceleration action, the velocity and acceleration along the longitudinal and lateral axis, the current steering angle and rate, as well as the angular velocity and acceleration.

C Failure cases

C.1 Roach

In this section, we show some examples of characteristic failure cases of the Roach reward function. The Roach model only achieves 22 DS on longest6 v2, so there are more failure cases than displayed here.

Another problem of the roach reward is the dependence on an A* algorithm for local path planning. The route that the agent should follow is computed by an A* algorithm. While this is, in principle, not a problem, the episode is also terminated if the policy drives 3.5 meters away from the lane chosen by the A* algorithm. This is a problem on multi-lane roads because it means the agent cannot decide which lane to drive on based on the traffic situation. Additionally, the A* algorithm lane changes at specific locations, which the agent has to adhere to. In case there is an oncoming vehicle in the target lane, the agent has to stop and wait for it to pass. Stopping at random locations on a highway is a dangerous policy that disrupts traffic flow and may lead to rear-end collisions. Additionally, we observe that a frequent failure case of Roach is that it does not stop, leading to a collision with an oncoming vehicle, when changing lanes at these predefined locations. This is illustrated in Fig. 4. Aside from RL agents, who inherit it, this is also a major limitation of SotA rule-based planners [97, 4, 5].

Fig. 5 shows the behavior of an early model after roughly 2 million samples. Roach waits at a green light, rendered as a green stop line. This is a very simple behavior that leads to high rewards since the agent will be given large speed rewards in future frames after the light turns red again. Unlike driving, waiting at a traffic light incurs no risk of collision.

Fig. 6 shows the behavior of a converged seed. Many training seeds eventually escape the behavior of waiting at green lights. They do, however, still slow down when approaching a green light, presumably trying to increase the chance of catching the next red light. Here, the model initially

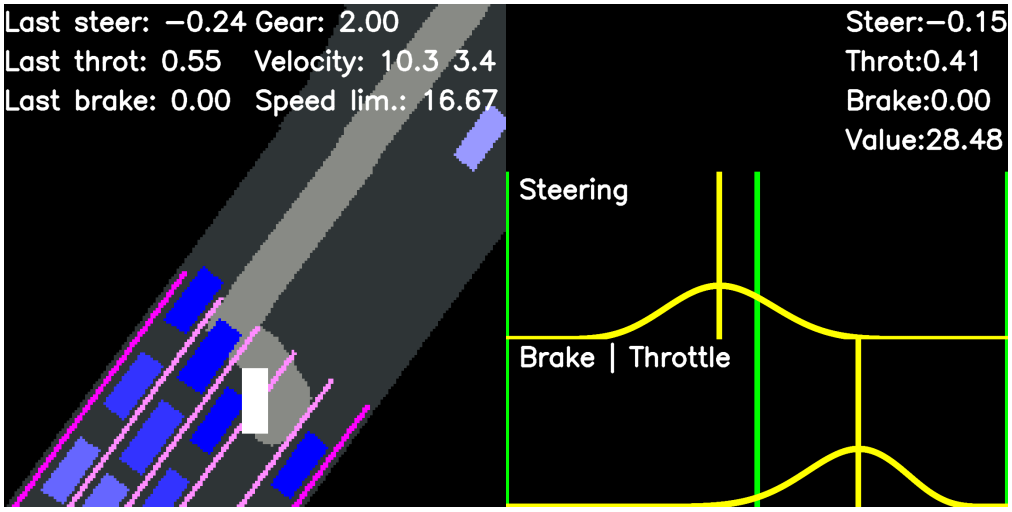


Figure 4: **Roach lane changes at predefined locations leading to rear collisions.** White is the ego car. Blue other cars. Lighter shades of blue represent past time steps. Gray is the precomputed route from the A* planner, and decides which lane to drive on.

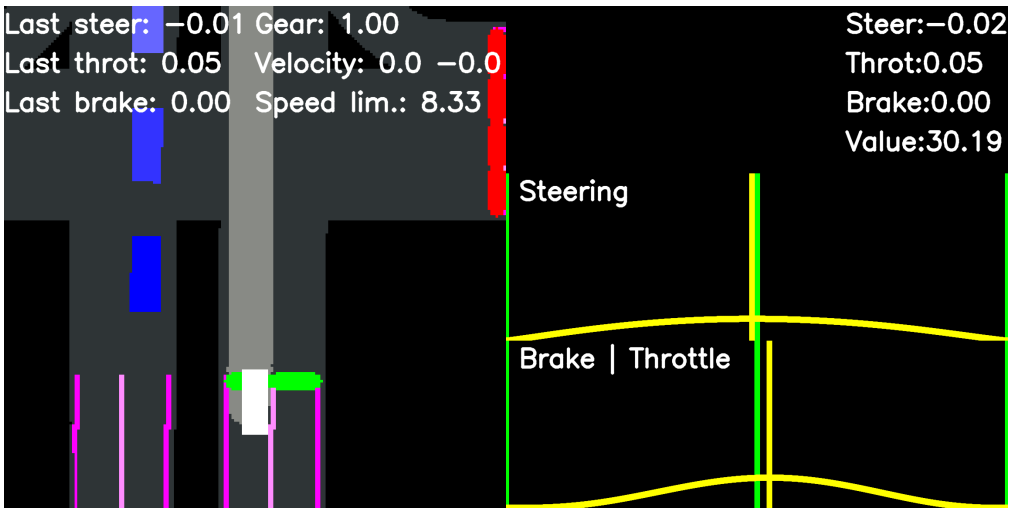


Figure 5: **Roach learns to wait at a green light early during training.** Note the 0 m/s velocity at the top of the image.

drives at 4 m/s, slows down to 3 m/s when it gets close to the traffic light. Once it passes the traffic light, the model starts accelerating again here to 5 m/s. Roach is trained via its reward function to drive at most 6 m/s.

C.2 CaRL

C.2.1 CARLA

Our final model, CaRL, has two common failure modes in CARLA, accounting for the majority of its infractions. First, CaRL often fails in the safety-critical scenarios that involve other cars running red lights. Fig. 7 shows an example where a non-reactive car is running a red light at high speed and crashes into CaRL from behind. CaRL tries to evade the opposing lane but accelerates too slowly to avoid the collision.

The second failure case is taking the wrong turn, most frequently missing highway off-ramps. An example is illustrated in Fig. 8 where CaRL continues driving on the highway instead of exiting on the left.

C.2.2 nuPlan

We show different failure modes of CaRL in Fig. 9. Avoiding collisions remains a challenge in nuPlan, partially due to the non-reactive agents or the abrupt behavior of reactive vehicles in the simulation modes. Specifically, pedestrians pose challenges to the planner. We estimate the small size and more flexible movement of pedestrians to be difficult for CaRL. Moreover, we point out that route completion is generally more difficult in the reactive nuPlan simulation, due to IDM vehicles blocking the road.

D Engineering

In supervised learning models are much larger than in reinforcement learning which has the consequence that most of the computation happens in highly optimized library functions. In RL in contrast models are comparatively tiny (we use a 2 million parameter model in this work) and most of the computation happens in user code. Since data is simulated and PPO scales well with more data (as shown in this work and others), this has the consequence that *the efficiency of the engineering has a direct impact on the performance of the model*, given a fixed time and compute budget. Doing (gradient) synchronized on-policy learning at scale poses special requirements to the simulation software, such as low loading times, fast time steps, and low variance between time steps. Standard evaluation simulation software is often suboptimal on one or more of these axes. In the following, we describe several changes we made to the CARLA leaderboard 2.0 and nuPlan simulation code (for training only) that improved our training throughput and enabled us to scale to 300 million and 1 billion samples. We are focusing on the most important changes since it is not possible to describe every little engineering detail in writing. The full details can be found in the code at https://github.com/autonomousvision/CaRL.

D.1 CARLA

Since we are training privileged models, we run CARLA in CPU mode using the "-nullrhi" option, which saves a lot of GPU memory. This feature was silently introduced in CARLA 0.9.14 alongside the multi-GPU rendering feature. The multi-GPU rendering feature decomposes the simulation into a CPU primary server and multiple GPU rendering servers. It is possible, as long as no sensors are used, to use zero GPU rendering servers, which effectively introduced CPU-only simulation.

In the CARLA leaderboard 2.0, the environment controls the agent, whereas in almost all RL codebases, the agent controls the environment. This makes applying the standard gym(nasium) interface [98, 99] hard. To resolve this, we split the environment and training code into two separate processes that communicate with each other via inter-process communication [100]. This incurs an overhead for sending the observations to the training process, but it is not significant compared to CARLA's expensive runtime. This enables us to use the gymnasium interface and build our code upon the CleanRL [101, 79] PPO implementation.

As documented in [27], the CARLA leaderboard 2.0 has very slow reset times that can be up to 1 minute. The reset consists of reloading the town, computing a new route with an A* algorithm, instantiating all cars and scenarios, and setting up the ego agent.

We speed up this process by specializing each simulator to a particular town. This avoids the need to reload the entire town assets and only requires destroying past cars and re-instantiating new once. PPO (Atari hyperparameters) uses 8 different simulators, which align nicely with the 8 small towns of CARLA (towns 1-7 and 10). Roach (CARLA hyperparameters) uses 6 simulators, we use town 1-6 which aligns with the evaluation towns of longest6 v2.

We remove the computation bottleneck of the A* algorithm by pre-computing it once for all of our experiments for a fixed set of randomly generated routes and storing the results on disk (see Section D.2). This increases RAM requirements, but modern servers have sufficient RAM capacity (our servers have > 100 GB per GPU).

When instantiating the agent, we do not reinitialize everything from scratch and only recompute values that are route-specific.

We train with the CARLA towns 1-7 and 10 and do not use the CARLA towns 11 - 13 or 15. This is because these towns are massive in size which significantly increases the runtime and reset speed of CARLA (and the reset can not be entirely avoided because only parts of the towns are loaded in memory at a time).

In this work, we have only modified the Python code of the CARLA leaderboard 2.0 client and used the original C++ simulation server. A particular problem with the CARLA server is that it has various bugs and crashes occasionally, particularly with the autogenerated scenarios we use. When scaling to over 100 concurrent CARLA servers, we observe that the occasional crashes become frequent crashes, as the probability that one of the 100 servers crashes increases. We monitor our training code for such crashes and restart everything if one occurs. We have tuned tight timeout values and optimized startup time to minimize the impact of these CARLA crashes. Nevertheless, the crashes are still a major bottleneck in our scaled-up runs, roughly doubling training time. A promising direction for future work to speed up RL training with CARLA further is to fork the simulator, fix the bugs, and optimize bottleneck functions like the slow CARLA HD-map retrieval.

We run the simulator at 10 Hz during training, which is the lowest frequency at which the CARLA physics are still accurate according to the documentation. During model evaluation, we run the simulator at the standard 20 Hz of the CARLA leaderboard 2.0 and use an action repeat of 2 for the policy. In preliminary experiments, we found that decreasing the simulator frequency was more efficient than using action repeat during training.

D.2 Scenario Generation

An important problem when training with the CARLA leaderboard 2.0 is that there are only a very small number of training scenario definitions available. These scenario definitions specify where a particular scenario type takes place and set the scenario hyperparameters. These scenario definitions were manually labeled, which is why they are limited in quantity. To generate the required number of samples for PPO with only the few available scenario definitions would require repeating the same routes many times and would likely lead to overfitting. Instead, we wrote an automatic route and scenario generation script based on rejection sampling and various heuristics. More principled solutions like self-play [68] or scenario generation methods [102, 103, 104] are not able to generate scenario types like construction sites, vehicle opening doors, or emergency vehicles required for the CARLA leaderboard 2.0. Extending these works to generate a wider variety of scenarios is a promising direction for future work.

The script works roughly as follows: First, we select a random point from the HD-map topology. We then pick the next point iteratively at 1-meter distance, choosing randomly at intersections. Outside of intersections, we change the lane with 10% probability. We generate more route points until the route is 1 km long. We run various checks to check if the generated route is valid, such as can the ego vehicle be spawned at the initial point. If not, we discard the route. Along the route, we then try to place scenarios of random types roughly every 100 meters. The scenario parameters are drawn from a random distribution, which is distributed according to how frequently the parameter occurs in the released handcrafted scenarios of CARLA. We then try to instantiate the scenario with the CARLA leaderboard 2.0 code and reject scenarios that fail to set up. If a generated scenario passes all validation checks, it gets added to the list of scenarios for that route. If not, we try another scenario type until all types have been tried. Various scenario types have special rules to account for the scenario's specific needs.

The resulting generated routes have up to 10 scenarios in them. The generated route and scenarios are noisy because our rejection tests are imperfect. Additionally, the number of scenarios generated may not be balanced because scenarios that can be placed in many locations are oversampled. Nevertheless, this script enables us to generate the large volume of unique training routes and scenarios needed to train PPO at scale.

On nuPlan, a large number of training scenarios are available, so generating scenarios was not necessary.

D.3 nuPlan

Dataset: The nuPlan recordings are stored in an SQLite database [105], whereas the maps are saved in a GeoPandas dataframe [106]. To reduce database interactions, we first collect the scenarios in the selected training set and store them in smaller pickle files with gzip compression. We additionally limit the number of entities at each frame by selecting the closest objects and agents to the position of the human operator. This is beneficial in nuPlan due to frequent scenes with 50+ pedestrians and cars, which slow down dataloading and rendering, effectively idling other parallel environments. To this end, we set a maximum of 50 vehicles, 50 pedestrians, 10 bicycles, and 10 static objects. Our final dataset uses all temporally non-overlapping scenarios (of 15s duration) with valid route information from nuPlan's training split, resulting in 269,075 samples and a storage size of 84 GB. Notably, we do not pre-process and cache any maps and provide the full planner input interface of nuPlan in the environment. We expect further performance could be gained from pre-processing the map and recordings, which would come at the cost of flexibility when computing the reward and observation of an RL agent.

Background Traffic: We directly integrate the reactive and non-reactive background traffic of the nuPlan simulator. In the non-reactive mode, the vehicles, pedestrians, bicycles, and objects are replayed as observed in logs. The reactive traffic simulates the vehicles along the centerline of the lane graph with an IDM lane following model [1], while the remaining agent categories are replayed from the logs. For training CaRL on nuPlan, we randomly select between reactive and non-reactive traffic during initialization with equal probability. We note that the reactive traffic is considerably slower, making it less adequate for short-cycled testing. We leave further optimization to future work.

Rendering: The observation rendering remains a considerable performance bottleneck (in addition to the reactive background traffic). In each iteration, we need to query nearby map elements from the map interface, retrieve relevant agents, and render several polygons and polylines in the respective channels. We use OpenCV for CPU-based rendering, which we found to be the fastest CPU-based library during development.

D.4 Hardware

Our code is CPU bottlenecked and highly parallelized, and benefits strongly from modern CPUs with many cores. We use servers with 12-24 physical cores per GPU for our experiments. Contemporary machine learning servers are primarily built around powerful high-memory GPUs for supervised training, often neglecting the CPU (with as low as 4 cores per GPU). We do not recommend such machines for our compute task since the low number of CPU cores means the compute job runs several times slower. Our jobs were run on expensive A100 and H100 GPUs simply because servers with high-end CPUs typically also have high-end GPUs. We do not fully utilize these expensive GPUs since our model is small. More economical GPUs, like the A6000, might suffice for our compute tasks.

E Baselines

E.1 CARLA

In this section we describe our reproduction of the CARLA baselines PlanT and Think2Drive.

E.1.1 PlanT

We implement PlanT [6], which is the state-of-the-art imitation-learning based baseline on CARLA. PlanT is a simple transformer-based planner that uses a sparse object representation for its inputs. All objects are input as bounding boxes with six attributes: x, y, yaw, speed, x-extent, and y-extent. Route boxes input the box ID instead of speed. The model outputs 4 waypoints for control using a GRU, which is additionally supplied with the target point and a traffic light flag.

Since the original PlanT model was trained using the leaderboard 1.0 version of longest6, we need to make some adjustments for a fair comparison. Most of these changes result from the increased ego speed, as described in Section 2.1.

Input: We add pedestrians and static obstacles as new object classes, following the representation of cars, since the original model did not consider them. Additionally, we add trigger boxes for ego-relevant stop signs and traffic lights to the object representation since the original flag-based approach was found to be ineffective at higher speeds and with the complexities of approaching and passing a stop sign. We only input red and yellow traffic lights and remove stop signs when the ego vehicle has slowed down enough to consider the stop sign as passed. Fig. 10 shows two training examples with the updated inputs.

Model: We keep the model architecture unchanged except for removing the traffic light flag in the GRU and adding the 4 new class embeddings. Since the maximum driving speed increases from 60 to 120 km/h, we double the number of bins for forecasting the other vehicles' speed from 16 to 32 to keep the bin size the same.

Control: To accommodate faster driving speeds, we replace the original control logic. We use the lateral controller implemented by PDM-Lite [5], using the interpolated waypoints as the target route and calculate the desired speed based on the distance between the second and fourth waypoints. Even though the sampling rate of our dataset is 4 Hz, twice that of the original paper, we keep the number of waypoints at 4. During testing, both increasing the number of waypoints to 8 and sampling the waypoints at 2 Hz showed poor performance, which may be caused by the limited amount of route information given by the route boxes, combined with the long distance forecasts resulting from higher driving speeds.

Training: We use a publicly available dataset collected using PDM-Lite [107], containing all 36 scenarios relevant to the leaderboard 2.0. At around 470k samples, it is similar in size to the 2x dataset used in the original paper. We follow the original training regime, training for 47 epochs with a batch size of 128 and a learning rate of 1e-4.

E.1.2 Think2Drive

In this section, we describe our reimplementation of Think2Drive [27], the code of which has not been officially released. Since several key hyperparameters are missing in the paper, we inferred them ourselves or used Roach [14] as a reference. Think2Drive is based on the model-based reinforcement learning algorithm DreamerV3 [52], combining a world model with an actor-critic approach.

Model: The Think2Drive paper does not specify which code version they have used. We tested both versions corresponding to the first [108] and second [52] DreamerV3 paper versions, finding that only the latter functioned properly. Consequently, we used the GitHub commit 251910d associated with the second paper version, which often outperformed the first version on most benchmarks,

Hyperparameter	Value
Hidden size	768
Recurrent units	6,144
Base CNN channels	96
Codes per latent	48
Reward loss scale	10.0
Replay buffer size	300,000
Parameters	125 Million

Table 13: Hyperparameters of our Think2Drive implementation. We used the second code version of DreamerV3 [52] and selected the 100M model to match Think2Drive's model capacity, while increasing encoder/decoder channels and reward loss scale similar to Think2Drive.

reported in the DreamerV3 paper. We selected the 100 million parameter model to match the model size used in Think2Drive and adjusted key hyperparameters accordingly. Table 13 shows the key hyperparameters used in our implementation.

Input: As input, we use the same BEV and scalar values as Think2Drive. Each image channel represents a binary mask for specific static elements and dynamic objects. For dynamic objects, we render previous locations (transformed to the current ego vehicle's coordinate frame) from four timesteps: current (t) and historical frames (t-5, t-10, t-15), each timestep being 0.05 seconds. Scalar inputs are sampled at these same timesteps.

Actions: We adopt the 30 discrete actions defined in the Think2Drive paper. Our simulator runs at 20 Hz while action selection occurs at 10 Hz, resulting in each action being repeated once (action repeat = 2).

Reward: Similar to Think2Drive, we use a weighted sum as our reward function:

$$r = 1 \cdot r_{speed} + 1 \cdot r_{travel} + 2 \cdot p_{deviation} + 0.5 \cdot c_{steer} \tag{2}$$

The speed reward r_{speed} encourages the agent to match a target speed, calculated as in (3).

$$r_{speed} = 1 - \frac{|v - v_{target}|}{7.5} \tag{3}$$

This target speed is determined by taking the minimum of: (1) 80% of the speed limit, (2) target speed considering the distance to the next red traffic light, (3) target speed considering the distance to the next uncleared stop sign, and (4) the target speed considering the distance to the next vehicle on the ego vehicles route. We linearly decrease the target speed given the distance to an obstructing actor, using (4).

$$v_{target} = 0.8 \cdot v_{limit} \cdot \frac{\text{clip}\{d - \text{safe margin}, 0, 12.5\}}{12.5} \tag{4}$$

The safe margin is 8.0, 4.0, and 2.5 meters for vehicles, traffic lights and stop signs respectively. For the travel reward r_{travel}, we use the distance traveled in meters. The deviation penalty $p_{deviation}$ penalizes deviations from the planned route as in (5).

$$p_{deviation} = -\frac{||p_{ego} - p_{route}||_2}{8.0} \tag{5}$$

where p_{ego} is the ego vehicle position and p_{route} is the nearest point on the route. We terminate the episode if the ego vehicle runs a red light, runs a stop sign, collides with any type of object, or deviates more than 15 m from the route. In case the ego has reached the end of the route up to 10 m or did not drive faster than 0.1 m/s for at least once during the last 100 s, we truncate the episode. For terminal rewards, we set the reward to -1 if the episode terminates due to exceeding the route deviation. In case of a collision, running a red light, or running a stop sign, we set the reward to $-1 - speed$ (where speed is the current velocity of the vehicle in m/s). If the vehicle reaches the end of the route, we assign a reward of $+1$.

Data Collection: We run 8 parallel CARLA instances (one for Towns 1-7 and Town 10), each generating 6 FPS, for a total throughput of 48 FPS.

Training: Following Think2Drive's approach, we implement curriculum learning over two training phases. First, we train 400,000 CARLA steps (800,000 in total, due to action repeat = 2) on basic routes with traffic lights, stop signs, and other vehicles. We then switch to the second training phase with 1.5 million CARLA steps on 1 km routes with scenarios along the route. We reinitialize both actor and critic parameters at 400,000 CARLA steps and subsequently every 800,000 steps throughout the second training phase. Following the first parameter reinitialization, we linearly increase the actor-critic train ratio from 16 to 64, which means each sample is ultimately used an average of 64 times for training. The world model maintains a constant train ratio of 16 throughout the training. This creates training iterations where only the actor-critic parameters are updated while the world model remains frozen. During these actor-critic-only updates, we prevent gradient propagation through the world model to avoid the critic's loss from dominating the world model's training dynamics. Our sampling strategy mirrors Think2Drive's approach: 50% samples are drawn uniformly from the replay buffer, while the remaining 50% come from the $K = 64$ frames immediately preceding episode termination.

E.2 nuPlan

This section provides additional baselines for nuPlan. All baselines were evaluated using the officially published code and model.

Metrics: The nuPlan closed-loop score (CLS) is assigned to each simulation clip and combines sets of multiplier penalties M and weighted metrics W.

$$\text{CLS} = \left(\prod_{m \in M} \text{score}_m \right) \times \left(\frac{\sum_{w \in W} \text{weight}_w \times \text{score}_w}{\sum_{w \in W} \text{weight}_w} \right) \in [0, 100] \tag{6}$$

The multiplier penalties include scores for no at-fault collisions (Coll.), drivable area compliance (Driv.), driving direction compliance (Dir.), and making progress (MP) to evaluate if the agent is stuck in a scenario. The weighted metrics contain scores for time-to-collision (TTC), route completion (RC), speed-limit compliance (Speed.), and comfort (Comf.). All scores are between 0-100, where higher values indicate better performance. The final CLS is an average over all scenario scores. We refer to [31] for further information.

Baselines: (1) The IDM planner is a rule-based baseline provided in the official `nuplan-devkit` repository[2]. The planner first retrieves a centerline path with a breadth-first-search on the lane graph and applies the IDM car-following model to control the ego-agent along the center path. (2) PDM-Closed is an extension of the IDM planner, that creates several trajectory proposals by combining different target speeds and centerline offsets. These proposals are simulated and scored with a similar metric to the nuPlan closed-loop score. (3) PLUTO uses a transformer-based architecture to forecast background agents and to regress multiple ego trajectories with corresponding scores. These outputs are then post-processed with the simulation and scoring modules of PDM-Closed to select a suitable ego trajectory, resulting in a hybrid planner. We use the code and checkpoint of the PLUTO repository[3] for reproduction. (4) Urban Driver applies PointNet layers on a vector representation of the scene with multi-head attention for aggregation and outputs an ego trajectory. (5) GC-PGP utilizes a graph neural network on the lane graph and agents to output multiple route-conditioned trajectories. (6) PlanCNN uses a CNN encoder on a semantic BEV rendering of the scene and outputs an ego trajectory with an MLP head. (7) PlanTF uses a Transformer architecture to encode agents and map elements in vector representation, forecast non-ego agents, and regress a single ego trajectory. The code and checkpoint are publicly provided in the official repository[4] (8)

[2] https://github.com/motional/nuplan-devkit
[3] https://github.com/jchengai/pluto
[4] https://github.com/jchengai/plantf

Diffusion Planner uses a DiT architecture to generate trajectories for ego and non-ego agents, while conditioning on a vector representation of the scene. Currently, the repository of Diffusion Planner [5] does not provide the guidance terms used for post-processing, but only code and checkpoints for purely learned planning. For PDM-Closed, Urban Driver, GC-PGP, and PlanCNN, we use the scripts and checkpoints from `tuplan_garage`[6].

Results: Table 14 shows the results on Val14 non-reactive traffic, while Table 15 displays the results on Val14 reactive traffic. As shown in prior work [4], the scoring mechanism of PDM-Closed improves IDM significantly. Rule-based and hybrid planners outperform all imitation learning based baselines. There is a particularly large gap in reactive traffic where the best open-source method, Diffusion Planner, still achieves 10 points CLS less than PDM-Closed. CaRL closes this gap to 1.5 points.

To achieve the best results in a metric, the global optimum of the optimization objective should be aligned with the global optimum of the metric [17]. The closed-loop score of nuPlan uses a weighted sum of 4 terms: route completion (RC), time to collision (TTC), speed limit compliance (Speed), and comfort (Comf.), which are traded off in a 5:5:4:2 ratio. While our reward works on both CARLA and nuPlan, it might not be optimal for this particular metric and its tradeoffs. We train another CaRL model, noted as CaRL (nuPlan tuned), where we modify the reward to reflect the tradeoffs of the nuPlan CLS and double the samples to 1 billion to maximize the CLS score. Instead of using multiplicative penalties for TTC, comfort, and speed, we add them to RC with the same ratios as used in the metric. This improves the score of CaRL by 2.6 points in non-reactive traffic and 2.5 points CLS in reactive traffic. This results in CLS of 93.87 in non-reactive and 93.12 in reactive traffic, marking the first time that a purely learning-based model outperforms the rule-based PDM-closed method in both traffic modes (the concurrent work GIGAFLOW [68] also outperforms PDM-Closed but only reports numbers for reactive traffic).

This reward introduces tradeoffs that introduce local minima that the optimization gets stuck in. When training with nuPlans (log-replay) traffic, this is not as much of a problem because the car is initialized at the (high) human speed, which means the model cannot avoid driving. Additionally, staying stationary can result in collisions with non-reactive traffic. This enables PPO to escape the local minimum of not driving (which gives optimal speed, comfort, and TTC rewards) early during training. We observed in preliminary experiments that training with this reward results in degraded performance in CARLA because of this problem. We therefore do not recommend this reward in general, but it is useful when the goal is to achieve maximum performance in nuPlan.

The trajectory of the human data annotator can be seen as an optimal for nuPlan, and it is often used as an upper-bound for the simulator. Most simulators, including nuPlan, have some unavoidable simulation errors. A perfect score of 100 is usually not possible.

We use the human trajectory here to demonstrate that nuPlans provided LQR controller, which is used in most nuPlan papers to convert the trajectory predicted by the planner to the actions of the car, is suboptimal. Table 14 shows that the performance of the human trajectory is 93.5 with the standard LQR controller of nuplan. We also report the log replay performance with an iterative LQR (iLQR), which is a more accurate controller that uses more computation. The iLQR improves the LQR controller by almost 3 CLS points, significantly raising the upper bound on nuPlan to 96.24. In particular, it improves collisions (Coll.) and drivable area compliance (Driv.). Additionally, we report the result of perfect tracking (Perf.), which perfectly tracks the human trajectory by teleporting the bicycle model. This raises the score by a small 0.15 and is the actual performance of the human driver (although the human does not necessarily follow the constraints of a bicycle model here).

We argue that simulators and benchmarks for autonomous driving methods should not make the control part of the simulation, as controllers are typically specific to a particular representation (here, trajectories) and can be suboptimal, as demonstrated here. The iLQR controller achieves great

[5] https://github.com/ZhengYinan-AIR/Diffusion-Planner
[6] https://github.com/autonomousvision/tuplan_garage

Method	Type	CLS ↑	Multiplier ↑				Weighted ↑			
			Coll.	Driv.	Dir.	MP	TTC	RC	Speed.	Comf.
IDM [1]	Rule	75.60	88.55	93.56	99.28	95.44	78.89	85.05	97.43	93.47
PDM-Closed [4]		92.84	**98.12**	99.64	**99.96**	99.11	93.29	92.14	**99.83**	95.08
PLUTO [12]	Hybr.	92.63	97.85	99.11	99.73	99.46	**94.36**	92.98	98.22	91.86
Urban Driver [109]	IL	50.42	69.14	79.25	95.21	97.67	61.99	91.50	81.75	**98.75**
GC-PGP [7]		59.60	84.97	88.73	98.39	89.80	79.25	60.19	99.31	89.00
PlanCNN [6]		70.81	87.66	91.95	96.74	94.54	82.38	83.81	97.86	87.39
PlanTF [8]		84.63	94.23	95.89	99.20	98.75	89.00	90.67	97.68	93.20
PLUTO* [12]		88.74	95.84	98.48	99.42	98.57	92.49	89.43	98.00	96.60
Diff. Planner* [32]		89.62	95.89	98.30	99.60	**99.91**	90.52	94.19	97.27	94.63
CaRL (Ours)	RL	91.25	97.36	99.28	99.02	99.37	92.04	94.44	98.74	89.09
CaRL (nuPlan tuned)		**93.87**	97.23	**100.00**	99.78	99.55	93.47	**96.14**	98.41	97.14
Log Replay (LQR)	*Human*	*93.53*	*98.79*	*97.94*	*99.06*	*100.00*	*94.28*	*99.00*	*96.50*	*99.28*
Log Replay (iLQR)		*96.24*	*99.55*	*99.55*	*99.24*	*100.00*	*96.51*	*99.32*	*96.46*	*98.93*
Log Replay (Perf.)		*96.39*	*99.78*	*99.46*	*99.37*	*100.00*	*96.78*	*99.28*	*95.86*	*99.20*

Table 14: **Performance with non-reactive traffic on Val14** (nuPlan)

Method	Type	CLS ↑	Multiplier ↑				Weighted ↑			
			Coll.	Driv.	Dir.	MP	TTC	RC	Speed.	Comf.
IDM [1]	Rule	77.33	89.22	93.02	99.24	96.33	81.57	85.20	97.20	93.11
PDM-Closed [4]		92.13	**97.90**	99.46	**99.96**	**99.11**	93.83	90.26	**99.83**	94.72
PLUTO [12]	Hybr.	89.66	97.09	99.28	99.87	98.57	**94.10**	86.11	98.86	91.86
Urban Driver [109]	IL	53.05	72.45	83.36	96.87	94.28	66.37	87.17	81.66	**99.28**
GC-PGP [7]		55.28	83.94	88.37	98.35	85.51	79.43	57.67	99.26	92.04
PlanCNN [6]		70.14	87.88	91.59	97.09	93.92	83.18	83.94	97.48	86.85
PlanTF [8]		76.14	95.21	96.33	99.33	89.53	90.61	77.21	98.50	93.56
PLUTO* [12]		77.97	96.69	97.59	99.69	87.66	93.92	74.56	98.70	95.62
Diff. Planner* [32]		82.73	93.07	97.85	99.82	96.42	88.19	85.88	98.29	89.45
CaRL (Ours)	RL	90.60	97.05	99.91	99.15	**99.11**	92.31	91.30	99.36	88.55
CaRL (nuPlan tuned)		**93.12**	97.09	**100.00**	99.91	**99.11**	93.29	**93.56**	99.17	97.05
Log Replay (LQR)	*Human*	80.32	85.64	97.94	99.06	100.00	79.79	99.00	96.50	99.28
Log Replay (iLQR)		82.05	85.69	99.55	99.28	100.00	80.77	99.32	96.46	98.93
Log Replay (Perf.)		82.02	85.60	99.46	99.37	100.00	80.77	99.28	95.86	99.20

Table 15: **Performance with reactive traffic on Val14** (nuPlan) *without post-prosessing

performance, yet is expensive, and giving researchers the flexibility to choose their approach for control can lead to innovative solutions that reduce this cost. In our work, we have merged planning and control into a single model which achieves both high performance and efficient inference.

As a side note, the human perfect tracking result might not be the best possible score one can achieve in nuPlan. Human drivers often intentionally exceed the speed limit by a bit to progress faster. The lowest auxiliary metric of the human driver is the speed limit compliance (Speed.). Almost all baselines are better at this metric, so it might be possible to achieve scores slightly above 96.4.

F Reward

This section describes the details of our proposed reward.

$$r_t = RC_t * \left(\prod p_t\right) - T \tag{7}$$

RC_t is the percentage of the route completed during the current time step.

F.1 Terminal Penalties

Our reward uses 6 terminal penalties, which lead to termination of the episode if violated.

Parameter	CARLA	nuPlan	Unit
Longitudinal acceleration	$(-20, 10)$	$(-4.05, 2.40)$	$\mathrm{m/s^2}$
Lateral acceleration	$(-9, 9)$	$(-4.89, 4.89)$	$\mathrm{m/s^2}$
Absolute Jerk	$(-30, 30)$	$(-8.37, 8.37)$	$\mathrm{m/s^3}$
Longitudinal Jerk	$(-30, 30)$	$(-4.13, 4.13)$	$\mathrm{m/s^3}$
Yaw rate	$(-1.0, 1.0)$	$(-0.95, 0.95)$	rad/s
Yaw acceleration	$(-3.0, 3.0)$	$(-1.93, 1.93)$	$\mathrm{rad/s^2}$

Table 16: **Comfort Thresholds.**

- **Collision:** The ego vehicle collided with any object.
- **Off road:** The ego vehicle drove off the drivable area.
- **Run red light:** The ego vehicle entered the intersection while its traffic light was red.
- **Run stop sign:** The ego vehicle crossed a stop sign without decelerating to 0 m/s first.
- **Route deviation:** The ego vehicle deviated from the route by more than 30 meters (wrong turn at intersections).
- **Blocked:** The ego vehicle did not move faster than 0.1 m/s for more than 90 seconds

The terminal penalty T is 1.0 for violating red lights and collisions and 0.0 for the other infractions.

F.2 Soft Penalties

We use the following soft penalties p_t that multiplicatively reduce the obtained route completion if violated:

- **Outside lanes:** The ego vehicle drives on the sidewalk or on lanes of the opposing traffic direction. $p_t = 0$
- **Distance to nearest lane center:** The ego vehicle is penalized by the lateral distance to the nearest centerline. The penalty factor p_t interpolates linearly from 1.0 while driving on a centerline to 0.0 while driving on a lane marking. Only applied outside intersections.
- **Speeding:** The agent exceeds the current speed limit. The penalty factor p_t is linearly decreased from 1.0 to 0.0 while the agent is between 0-8 km/h too fast.
- **Time to collision:** If the agent violated the time to collision (TTC) metric. $p_t = 0.5$
- **Comfort:** The agent is penalized if it exceeds certain comfort thresholds. There are 6 comfort metrics described below. Each reduce the penalty by $p_t = 1.0 - 0.5 * \#/6$ with $\#$ being the number of violated comfort constraints.

The penalty "Distance to nearest lane center" is there to encode the purpose of lane markings. Without it, the agent still learns to drive but does not know about the concept of lanes, leading to an unnatural-looking driving style that only considers drivable area. Speed limits are initialized at 30 km/h when vehicles are spawned in CARLA and updated once the vehicles pass the next speed sign. The comfort bounds in CARLA are shown in Table 16.

Comfort penalties and TTC if violated are applied for the next 500 simulator steps in CARLA and until the end of the episode in nuPlan. All other soft penalties are applied for one simulator step. For the CARLA vehicle physics, there are no human reference bounds for comfort, which is a subjective feeling and hence needs to be tuned based on data. We set quite loose bounds to ensure the agent can adhere to them, but future work may tune the bounds to achieve more comfortable policies.

Time to collision (TTC) is computed by performing a constant speed forecast of all actors in the scene. We forecast for 1 second, discretized to 5 200 ms intervals, with a kinematic bicycle model. TTC is violated if the ego bounding box intersects with another bounding box in the forecast.

F.3 nuPlan Specific Adaptations

Terminal Penalties: (1) **Collision**: We terminate the simulation if the bounding box of an ego vehicle intersects with another agent or static object. We ignore collisions if the ego vehicle is stationary (i.e.has a speed of < 0.05 m/s), due to some unavoidable collisions with non-reactive agents. (2) **Off road:** The drivable area in nuPlan is defined by the polygons of the road and car-park areas. If the ego agent leaves the drivable surface, we terminate the simulation. We do not use the terminal penalties for (3) **Red light** and (4) **Stop sign**, as they are not accounted for in the nuPlan CLS and pose several challenges for implementation. For example, the human vehicle operator regularly drives over red lights and stop sign polygons, due to label noise and human behavior, making an optimal score not feasible. Similarly, we ignore the (5) **Route deviation** and (6) **Blocked** penalties, as it is hard to select a distance or duration threshold for these penalties in the short durations of the nuPlan simulation (i.e. 15 seconds).

Soft Penalties: (1) **Outside Lanes**: We apply a penalty of $p_t = 0$ if the ego agent leaves the route polygons, which coincides with driving in the opposite traffic or car park areas. For (2) **Distance to nearest lane center**, we allow a deviation of 0.5m (i.e.$p_t = 1.0$) from the lane center and linearly decrease the penalty from 1.0 to 0.0 starting from 0.5m to the lane marking. Allowing some center deviation was necessary, as driving in the middle of the lane can be suboptimal in nuPlan (e.g.if cars are parked on the side of the road). The (3) **Speeding**, (4) **TTC**, and (5) **Comfort** penalties are defined as in Section F.2. Note that we use stricter comfort boundaries, as shown in Table 16, which are the same thresholds used in the nuPlan comfort metric and selected based on human trajectories from the dataset [31].

Survival Bonus: Due to the short training scenes in nuPlan, it is then possible to reach 100% of the route completion before the simulation ends. In this case, the reward from (7) does not provide feedback to the ego agent and no incentives to avoid terminal penalties. We found that PPO planners would behave aggressively to foster short-term progress while putting less of an emphasis on collision avoidance. We circumvent this problem by combining the regular reward with a constant survival bonus, as follows

$$r_t^* = (1 - s) \times r_t + \frac{s \times 100}{N}, \tag{8}$$

where $s \in [0, 1]$ is the survival ratio and N the total number of simulation iterations (i.e. 150 for 15s at 10Hz). The weighted combination ensures the sum of rewards remains in the range of 0-100, as previously. We use a survival ratio of $s = 0.6$.

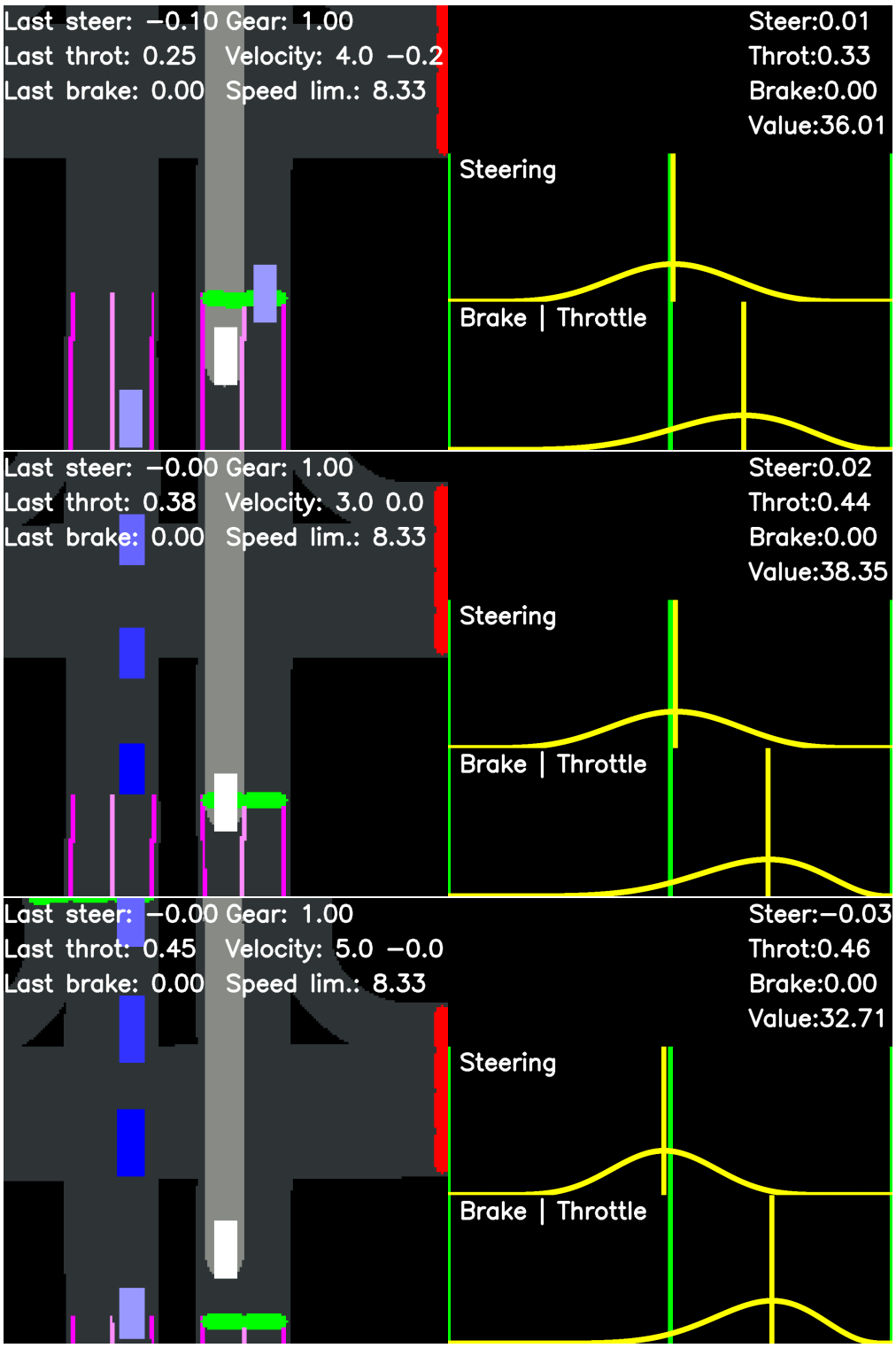


Figure 6: **Roach slows down at green lights.** Top to bottom are 3 different time steps. Note how the velocity initially slows down from 4 m/s to 3 m/s. Once the agent passed the green light, the model started accelerating again to 5 m/s.

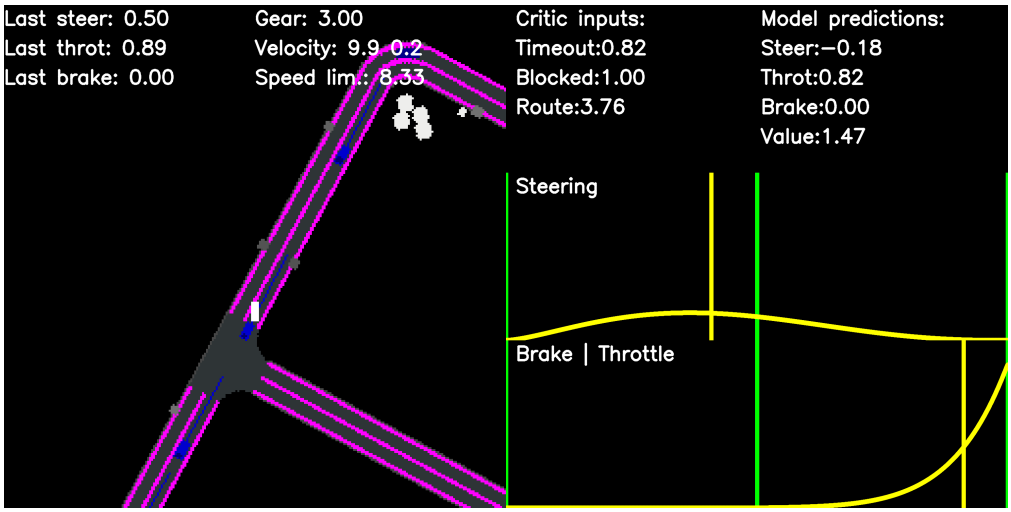
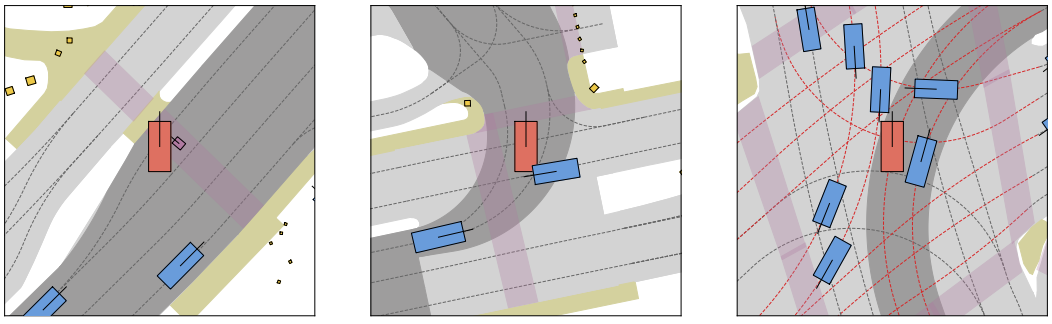


Figure 7: **Another car runs a red light and rams the CaRL from behind.**

Figure 8: **CaRL misses a highway exit.**



(a) Pedestrian collision. (b) Vehicle collision. (c) Ego stuck in traffic.

Figure 9: **Failures of CaRL in nuPlan.** (a) The ego vehicle collides with non-reactive pedestrians and veers off-road in an attempt at collision avoidance. (b) During an unprotected turn, the ego vehicle has a rear-side collision with a non-reactive vehicle. (c) Non-ego vehicles regularly block the road in the reactive simulation, resulting in low progress scores.

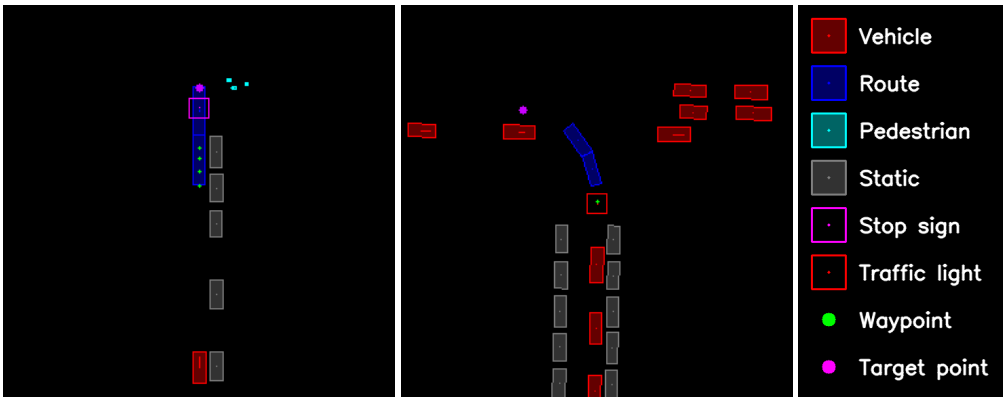


Figure 10: Training examples with the updated PlanT inputs. The image on the left shows the ego vehicle approaching a stop sign with pedestrians crossing the road. The second example shows the ego vehicle waiting to turn left at a red light. The ego vehicle is always at the center of the image.