End-to-end Methods for Autonomous Driving in Simulation

Aaron Huang

CMU-RI-TR-21-65

August 5, 2021

The Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA

Thesis Committee:
Prof. Deva K. Ramanan, *chair*
Prof. John M. Dolan
Prof. Kris M. Kitani
Dr. Peiyun Hu

Submitted in partial fulfillment of the requirements for the degree of Master of Science in Robotics.

To my parents - I am eternally grateful for your love and support.

Abstract

Fully autonomous driving is considered one of the grand challenges of modern technology and a variety of approaches have emerged for creating and evaluating autonomous driving agents. The self-driving industry typically adopts a modular software architecture and uses large fleets of autonomous vehicles for data collection and evaluation. *Simulation* generally occupies a minor role in these workflows as a tool for basic unit tests and debugging. However, recent end-to-end methods in academia have elevated simulation's role in the training of machine learning models and demonstrate state-of-the-art performance on simulated driving benchmarks as a result.

In this thesis, we consider two questions. First, what are the state-of-the-art approaches for end-to-end driving in simulation and how can we improve them? We use the open-source CARLA simulator as our testbed and work with the recent Leaderboard benchmark for evaluation. In our first project, we give a detailed failure analysis of "Learning by Cheating" (LBC) which has been considered the state-of-the-art in imitation learning. We then propose *offline imitative reinforcement learning* (OIRL) as an alternative approach and investigate its performance. In our second project, we give a failure analysis of the "World on Rails" (WOR) offline reinforcement learning method and propose "Slightly Online Reinforcement Learning" (SORL) as an improvement.

Second, what value does simulation provide for the training and evaluation of driving algorithms? We draw attention to the crucial role that *privileged information* plays in both LBC and WOR's algorithms. We emphasize that these methods are *impossible* to replicate without simulation and advocate for privileged information as an important tool for developing planning algorithms. Finally, we give a critical analysis of CARLA by pointing out issues in its perceptual and behavioral realism. We also critique the Leaderboard benchmark and provide discussion on why its primary metric cannot be reliably used as a proxy for good driving behavior.

Acknowledgments

I would like to first thank my advisors, Deva Ramanan and John Dolan, for the guidance they have given me during my two years at the Robotics Institute. I appreciate the unique perspectives and especially the critical support each of them provided as we adjusted to working from home amidst the global pandemic and the significant changes to my research projects that followed. I go into the next phases of my career as a more confident and versatile researcher because of them.

I want to express heartfelt gratitude to Peiyun Hu for not only serving on my thesis committee and being a great collaborator, but for also being a true friend and mentor in my journey through graduate school. I will always think back fondly on the insightful chats we had about research and life! I am also grateful to William Qi for being one of my first friends at school and for Sean, Krishna, Tarasha, Martin, Jason, Ravi, Zhiqiu, and everyone else in Deva's lab for being amazing and supportive lab-mates.

I also want to thank Kris Kitani for serving on my thesis committee and giving constructive feedback on my research over the last year. Besides that, I'm incredibly appreciative of how much effort and preparation he put into teaching one of my classes - he truly cares about helping students learn and is a role model for how to be a great teacher.

I'm also especially grateful to Kartik, Mayank, Sachit, Sanil, Barath, and all the other MRSD folks for being my closest friends at school. I'll always remember late hours spent working on homework together and the occasional weekend get-togethers at my apartment!

And finally, I am forever indebted to my parents - all of the opportunities and successes I have experienced are only possible because of their hard work and perseverance as first-generation immigrants in America. I am eternally grateful for your ceaseless love and support.

Contents

When this dissertation is viewed as a PDF, the page header is a link to this Table of Contents.

Chapter 1

Introduction

1.1 Motivation

Autonomous driving is one of the grand challenges of modern technology, and its successful development would universally benefit quality of life and public safety. For example, the modern transit system is defined by high rates of private car ownership which is an incredibly inefficient mode of transportation. Commuter cars are typically only used for an hour or two every day, yet they require infrastructure like parking garages and large roads to be built while introducing gridlock and long commute times in densely populated areas. In 2015, the average resident in Los Angeles and Sydney spent seven working weeks each year commuting with many of those hours spent in standstill traffic [11]. This is not only a waste of time, but also comes at very real cost to personal health and safety. A recent study suggests that workers who commute more than 10 miles a day are more likely to have elevated blood pressure [14]. Tragically, more than 30,000 people die from driving accidents every year in the United States while over 4 million are injured seriously enough to require medical attention, with a recent NHTSA study suggesting that 94% of accidents are a result of human error [31].

In recent years, many self-driving companies and start-ups have raised billions of dollars in the hopes of developing autonomous robotaxis that drive more safely than humans while breaking our addiction to private car ownership. However, despite more than a decade of strenuous efforts, we are still a long way from realizing this

vision of autonomous transportation. Driving is a difficult task that requires fast decision-making in complex and confusing scenarios, with tragic consequences if done poorly. Current state-of-the-art systems are extremely good at handling the majority of simple driving scenarios like lane-keeping and merges. The crucial component is showing that they can robustly generalize to novel "long-tail" scenarios that emerge infrequently, but cause the majority of driving accidents [23]. Long-tail scenarios are (by definition) underrepresented in real world driving data, but *simulation* offers a way for roboticists to test their software in dangerous situations without any actual risk. However, simulators in industry are typically used for unit-testing and virtual evaluation and play no role in the actual training of decision-making algorithms.

Recent progress in artificial intelligence research has produced impressively capable decision-making agents, with simulation playing an increasingly key role. Machine learning techniques in particular have demonstrated strong performance on many complex tasks [15, 24, 30]. These sophisticated methods can learn strong policies given high-dimensional inputs (sometimes from multiple modalities) and sparse performance cues. For autonomous driving, there are a plethora of learning-based approaches that are categorized based on modularity. Highly modular approaches decompose driving into subtasks, with one common split being perception, prediction, planning, and control. Machine learning models are often used in perception and prediction, but they are usually fine-tuned for their specific task and are not jointly trained. Modular approaches are adopted by most self-driving companies, but they tend to be difficult to implement and manage. On the other hand, end-to-end learning methods set up models that learn direct mappings from sensory inputs to driving controls. These methods are popular in academia because they are relatively easy to implement and have already shown promise in other domains.

1.2 Contributions and Organization

Very broadly, we consider the following questions: (1) what is the current state-of-the-art in simulated autonomous driving, (2) what value does simulation provide for the development of these algorithms? We believe that simulation enables unique, promising learning-based approaches that would be expensive or even impossible to train without simulation. However, there are still many issues with open-source

simulators limiting their usefulness for training and evaluation. Our contributions and organization are as follows:

- In Chapter 2, we introduce the CARLA driving simulator and the Leaderboard benchmark used for all experiments and results. We describe our contributions to the Leaderboard evaluation toolkit that record additional metrics and generate new visualizations, providing actionable feedback at a glance. The augmented toolkit enables highly detailed failure analysis of Leaderboard driving methods.
- In Chapter 3, we use the augmented toolkit to provide an extensive failure analysis of the "Learning by Cheating" baseline. We introduce OIRL, a hybrid reinforcement and imitation learning approach inspired by the "Deep Q Learning from Demonstrations" method.
- In Chapter 4, we use the augmented toolkit to provide an extensive failure analysis of the "World-on-Rails" baseline. We introduce SORL, a middle-ground approach to offline and online reinforcement learning inspired by DAgger.
- in Chapter 5, we critique CARLA and Leaderboard on their perceptual and behavioral realism compared to real-world driving and discuss issues with the "driving score" metric used to judge driving agents. We conclude by advocating for simulation to play a more significant role in developing robotics algorithms.

Chapter 2

CARLA Simulator

2.1 Background

Simulators for autonomous driving (and robotics in general) have long been low-quality proof-of-concept testbeds. TORCS is an early example that simulates racecars driving around a track [12]. However, *realistic* driving simulators are a recent phenomenon. Many simulators have been developed by self-driving car companies, but these are typically not available for public use. In 2017, the open-source CARLA (CAR Learning to Act) project was announced with the goal of creating simulated environments for the training and evaluation of autonomous driving agents [10]. Despite being a relatively new project, CARLA has grown to include an impressive set of features and is the de-facto simulator for the AV research community. In this chapter, we will review CARLA's recent driving benchmarks and available features [1].

2.2 Features

2.2.1 Environments

CARLA environments are designed to replicate a variety of urban, suburban, and highway environments with high visual and behavioral fidelity. These towns present a diverse set of driving situations including highway merges, busy traffic intersections, and local single-lane roads. Each town is populated with 3D models of static objects

like buildings, traffic infrastructure, and dynamic objects like pedestrians, cyclists, and vehicles. These models are rendered using Unreal Engine 4 and a library of assets commissioned by the CARLA team, which were created with diversity and realism in mind. For example, many real-world car models like the Tesla Cybertruck and Model 3 appear in CARLA, and pedestrians are rendered with a variety of outfits and accessories.

Traffic simulation is another important component of accurate simulated driving. Non-player vehicles in CARLA are controlled by the TrafficManager, which aims to populate the simulation with realistic traffic conditions. The TrafficManager runs each vehicle under its command in autopilot mode, which uses hard-coded rules-based logic to navigate. Crucially, the autopilot uses privileged information, which is defined as any information that a real-world driver would not have access to. This includes things like true distance to surrounding objects, traffic light state, and other agents' navigation controls/long-term goals. Autopilot planning is separated into a local and global stage. Global planning creates high-level driving routes for AI vehicles to navigate at a "Google Maps" level of fidelity, while local planning produces the actual vehicle controls and handles more immediate issues like obstacle avoidance and traffic rule adherence. Unfortunately, pedestrian behaviors are not as well-modeled, but they can be given simple paths to follow. However, they will not exhibit any interest in self-preservation, unlike real people.

2.2.2 Sensors

CARLA comes with an impressive sensor suite that gives agents access to rich sources of perceptual information, which is crucial for most learning-based approaches. Developers can attach multiple sensors to any agent in the environment and replicate the full sensor stack for a typical autonomous vehicle. Importantly, each sensor has an assortment of attributes that can be configured to the developer's specification. For example, RGB cameras can have shutter speed, capture rate, field of view, depth of field, and many other attributes specified. LiDAR sensors can have the upper/lower field of view, dropoff rates, and number of channels specified. One unique advantage to working in simulation is access to *privileged information*. While this was previously discussed in the context of planning, CARLA includes two privileged sensors: semantic

segmentation cameras and semantic LiDAR sensors. These sensing capabilities are difficult or downright impossible to replicate in the real world, but allow for unique training setups in simulation. For example, training semantic segmentation algorithms in the real world is expensive and time consuming due to the need for manually-annotated segmentation masks. On the other hand, you can create an essentially limitless amount of similar data in CARLA using the semantic segmentation camera. As we will later see, the "Learning by Cheating" approach directly uses semantic segmentation camera images as input for its strong expert.

2.3 Benchmarks

2.3.1 NoCrash and CoRL2017

In the original CARLA paper, the authors create the "CoRL2017" experiment suite as the first benchmark for driving in CARLA [10]. The CoRL2017 suite contains 24 experiments that evaluate goal-directed navigation and include four basic tasks. The first two tasks are basic drive straight and single turning tasks with no other agents present. The third task is a general navigation task with no restriction on the location of the goal, while the fourth task adds cars and pedestrians to the third task. An evaluation episode is considered a success if the candidate agent reaches the goal within a specific time budget. The time budget defined as the amount of time needed to reach the goal along an optimal path while driving at 10 km/h. There are six weather conditions and two towns available, and the full benchmark evaluation runs every combination of task, town, and weather for 25 episodes. Infractions like driving on the sidewalk and collisions are recorded, but they do not terminate the run.

The CARLA team later introduced the larger scale "NoCrash" benchmark with the goal of evaluating how agents handle complex events that are caused by changing traffic conditions and dynamic agents [9]. Similar to CoRL2017, there are three goal-directed navigation tasks where the agent starts at a random position and is directed by a high-level planner towards a goal. The three tasks are: empty town, regular traffic, and dense traffic. Each task is run for 25 episodes and repeated for each combination of six weather conditions and two towns. While CoRL2017 did not terminate evaluation episodes upon infractions, NoCrash will end the run if the

candidate agent has a collision that is larger than some predefined magnitude. An episode is considered a success if the agent reaches the goal within the time limit, and without any major collisions. NoCrash also reports the percentage of traffic light violations as a secondary metric.

2.3.2 Leaderboard

Leaderboard is the most recent official benchmark released by the CARLA team, and is one of the primary benchmarks for autonomous driving evaluation in academia [2]. Leaderboard provides two contributions - a public evaluation server for standardized benchmarking, and a high-quality toolkit with many useful capabilities. We use Leaderboard as the primary testbed for all work in this thesis.

Leaderboard evaluates driving agents based on how well they drive on a held-out set of routes that are hosted on the evaluation server. There are additionally 50 training routes and 26 validation routes that have been made available to the public. The agent must navigate each route according to a set of coarse waypoints, each of which has a GPS coordinate and a high-level command like "go-straight" or "turn-left". These waypoints are not dense and are often separated by hundreds of meters, so agents must compute local plans that progress along the route while also avoiding infractions. The benchmark is scored based on a primary metric called driving score, which is computed from a secondary route completion score and an infraction penalty as described below. Unlike NoCrash, major collisions do not immediately terminate a run - instead, infractions accumulate an infraction penalty multiplier that is applied to the route completion score, yielding the final driving score.

- **Driving score**: $\frac{1}{N}\sum_{i}^{N} R_i P_i$ - primary Leaderboard metric which determines the official benchmark standings. N is the total number of routes tested, with R_i and P_i respectively denoting the route completion percentage and infraction penalty of the i-th route.
- **Route completion**: R_i - route completion percentage of an agent on route i
- **Infraction penalty**: $P_i = \prod_j p_i^j$ - aggregates all penalties incurred during route i. Agents begin at an ideal score of 1.0, which is reduced by a penalty multiplier for each infraction instance (e.g. 0.5x for a pedestrian collision, 0.8x for running a stop sign).

One major feature that Leaderboard introduces is scenario crafting and injection. In an effort to create more realistic and challenging evaluations, Leaderboard comes with pre-packaged scenarios that are inspired by the National Highway Traffic Safety Association (NHTSA) pre-crash typology of scenarios [23]. At test time, scenarios are triggered when agents reach specific locations along each route, necessitating an immediate response to the situation. For example, the "VehicleCrossingRoute" scenario occurs right after the ego agent turns at an intersection, spawning a pedestrian that runs directly across the road in front of the ego vehicle.

There are two Leaderboard tracks that represent the two dominant approaches for autonomous driving in industry. The SENSORS track gives agents access to a predetermined set of common sensors like RGB cameras, LiDAR, and RADAR. There are limits on the number of each sensor agents are allowed to use and constraints on their physical placement on the simulated test vehicle. The MAP track gives access to the same sensors as the SENSOR track, with an additional map "sensor" that provides high-level road network information during deployment. This resembles the high-definition static maps that autonomous vehicles use to localize and resolve ambiguity.

In addition to the benchmark, the Leaderboard codebase was made open-source to developers. On each route deployment, Leaderboard's evaluation toolkit keeps track of the ego agent's route completion score, a list of infractions that the agent caused and where they occurred, and the final driving score. This information is written to a log that can be manually inspected after evaluation. However, this approach to failure analysis is time-consuming and frankly quite unrevealing. This workflow requires exhaustive sifting through tens or hundreds of JSON-formatted log files, with no context about how each particular infraction might have arisen aside from a 3D coordinate of where it occurred. In other words, the available feedback is extremely coarse and does not help developers identify and address the weaknesses in their methods.

2.4 Augmented Evaluation

We contribute to the toolkit by modifying the codebase to produce four new visuals that enable comprehensive and actionable feedback at a quick glance. This required

modification to the way Leaderboard bookkeeps infractions and route completion scores. We show an example of each visual, and how it can be used to drill down into any Leaderboard method's weaknesses and failure modes.

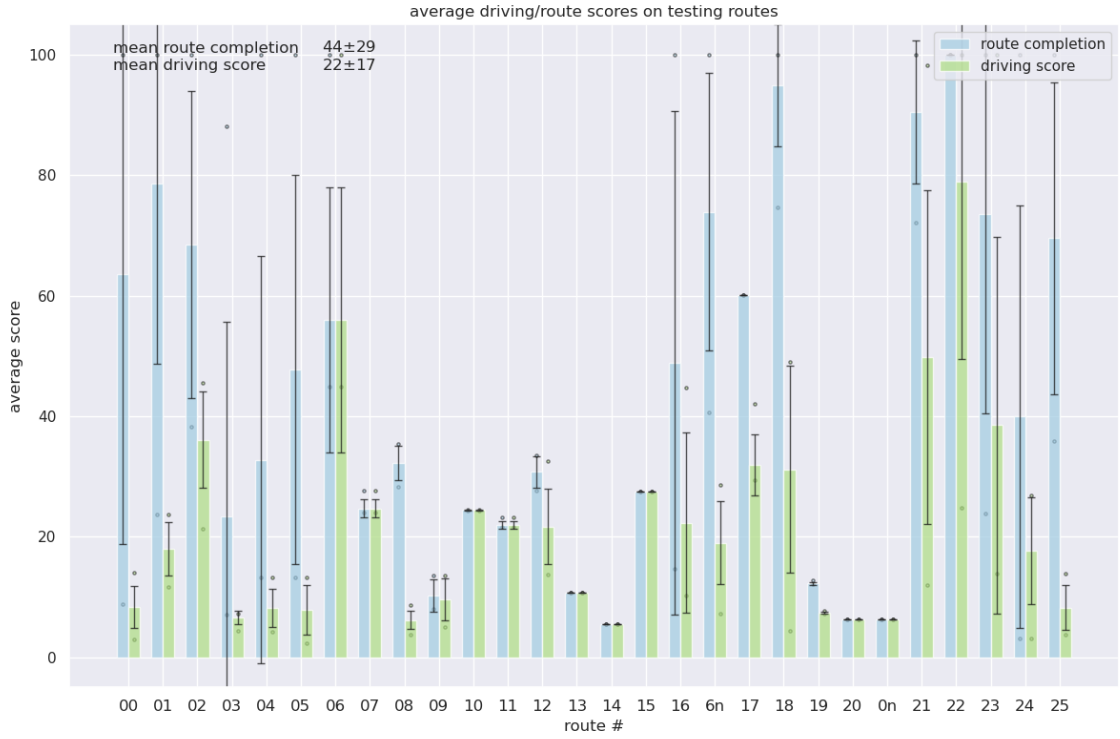


Figure 2.1: Average route metrics over all routes

Figure 2.1 gives a "bird's-eye-view" picture of driving performance by agglomerating the primary statistics over all routes. Since each route is run multiple times, we plot the mean value of driving score and route completion with error bars indicating standard deviation. The two open dots in each bar show the maximum and minimum scores achieved. This plot lets developers identify routes that their models struggle with and go deeper with the failure analysis with the other available plots. For example, routes 8, 9, and 14 consistently yield poor driving scores for this particular method while routes 6 and 23 have highly variable route completion scores, implying that we should perform additional failure analysis on those routes.

Figure 2.2 also gives a bird's-eye-view analysis, this time focusing on infractions incurred over all routes and repetitions. The pie chart shows the distribution of

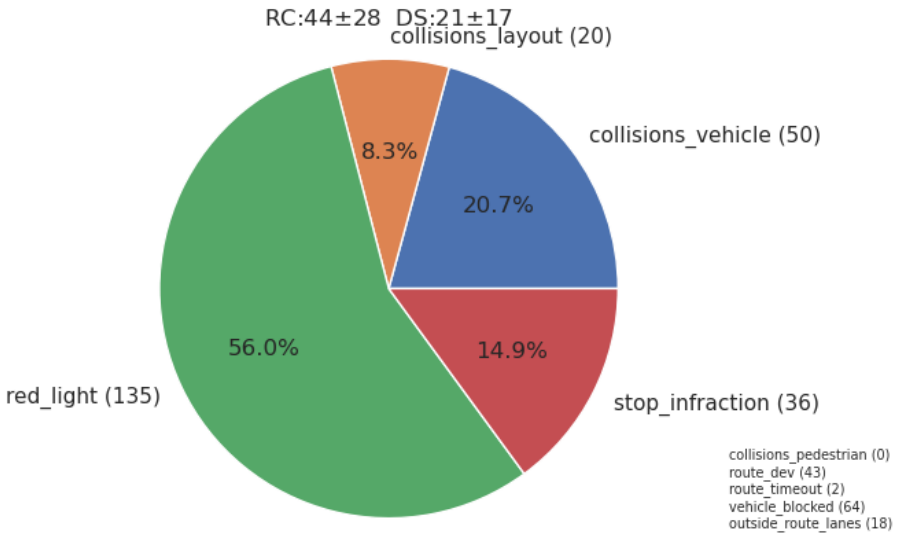


Figure 2.2: Infraction pie chart showing total infractions over all routes/repetitions

infractions that incur a penalty multiplier. The text on the bottom right shows infractions that either did not occur at all over the runs (e.g. pedestrian collisions), or terminate the run (with the exception of "outside route lanes" infractions). One thing to note is that some infractions are often a trigger for other infractions - for example, running a red light typically puts the ego-agent in a situation where it is more likely to hit a vehicle or deviate from the route. In that case, it would be prudent to focus even more on red light infractions. While this infraction plot gives a good general sense of what infractions a method struggles with, more granular analysis of qualitative examples is required for identifying causes and possible solutions.

Figure 2.3 considers the infractions that an agent incurred on all repetitions of a route, and plots their average values as a bar chart. For a set of 26 validation routes, 26 plots will be created. As in the first chart, the bar is plotted at the mean with error bars indicating standard deviation, and open dots indicating maximum and minimum values over all repetitions. This chart makes it significantly easier to identify which routes cause specific types of infractions. For example, if we would like to take a deeper look at routes that have collisions with static objects, we can just quickly look through the 26 plots and pick out the relevant routes. In this case, we can identify a variety of penalty infractions on this route like stop sign/red light running, and

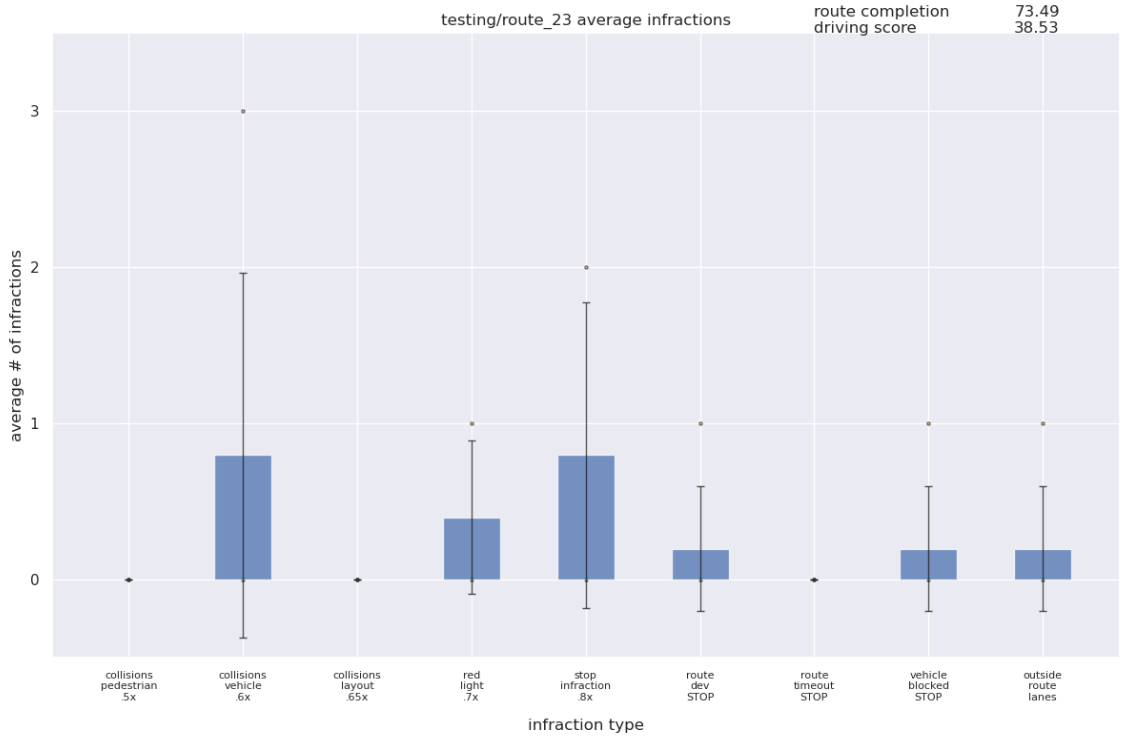


Figure 2.3: Average infraction counts over all repetitions of route 23

terminating infractions like route deviations and blocked agent timeouts.

Figure 2.4 affords the most granular level of failure analysis. For every repetition in a route, this plot shows the specific progression of driving score and route completion over time. The red dotted vertical lines show timestamps where an infraction occurred, with the specific infraction type and penalty multiplier displayed in the top text. The purple dotted vertical lines show timestamps where a scenario was triggered, with the specific scenario type displayed in the bottom text. For example, the agent hits a vehicle at 6 minutes 50 seconds into the run directly after running a stop sign. It is best practice to generate evaluation videos, as these can be used in conjunction with these plots to directly view a particular infraction and determine causes and possible solutions.

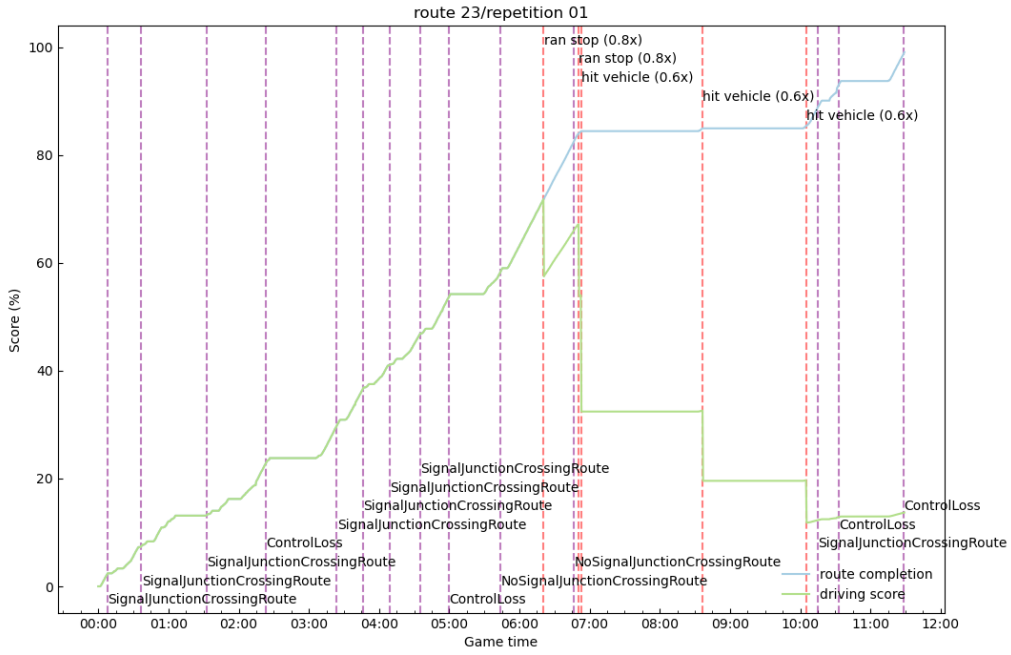


Figure 2.4: Driving score and route completion performance over time for repetition 1 on route 23. Red lines show infractions with text indicating infraction type and penalty multiplier. Purple lines show Leaderboard scenarios.

Chapter 3

Offline Imitative Reinforcement Learning

3.1 Background

3.1.1 Imitation Learning

Imitation learning is a classic technique for end-to-end driving approaches. Behavior cloning is the most basic form of imitation learning and refers to methods that learn a direct mapping from observations to actions using expert demonstrations. ALVINN [26] is the classic example of behavioral cloning which uses a neural network to learn mapping from image to steering angle, and demonstrates road-following along a forest road. More recently, [5] used a convolutional neural network to show real-world vehicle control in a bevy of different driving scenarios. [6] uses video game driving demonstrations to train a network that maps images to affordances that can be directly used for control. [8] proposes conditional imitation learning (CIL) which trains a network to predict steering and acceleration commands from images while conditioning on a high level-command. Aside from introducing the NoCrash benchmark, [9] updates CIL by proposing CILRS which is likely state-of-the-art in single-stage behavior cloning. More recently, [7] proposes a two-stage approach to behavior cloning. Their results suggest that a privileged imitative learner, despite performing worse than the expert it learns from, may serve as a better teacher to

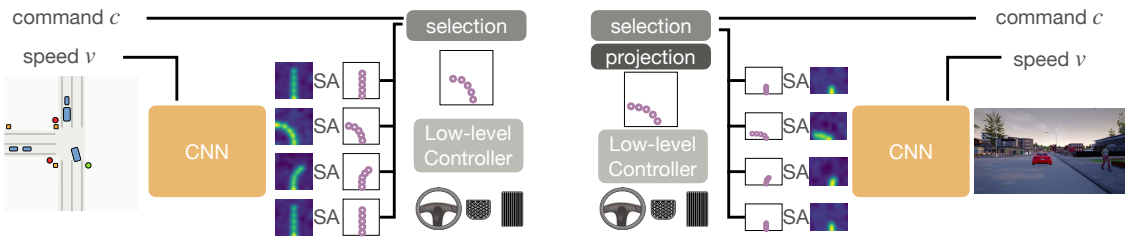
non-privileged imitative learners, by providing richer supervision.

3.1.2 Reinforcement Learning

Reinforcement learning (RL) is a well-studied field that has traditionally been applied to decision-making and control problems. We point to the classic treatise written by Sutton and Barto as a review of the fundamentals of RL [32]. More recently, the deep Q network (DQN) work by DeepMind is arguably the most seminal RL work of the last decade [22]. They pioneered the use of deep convolutional networks to train agents that demonstrate superhuman performance on a variety of Atari games. DQN is considered an off-policy algorithm, which means that it trains on data that is not collected by the learning agent. Besides DQN, DDPG is considered a foundational off-policy method with TD3 being a more recent incarnation [13, 21]. In the realm of simulated autonomous driving, [4] show perfect performance on the CoRL2017 benchmark and show strong performance on NoCrash using the double DQN approach. In contrast, on-policy algorithms train on data that is gathered by the learning agent, and does not train on old data. The classic vanilla policy gradient algorithm is a foundational on-policy algorithm, with TRPO being a more recent approach [28, 33]. [3] uses the PPO method [29] to again show perfect performance on CoRL2017 and stronger performance on NoCrash. However, both [4] and [3] use *privileged information* like true distance to nearby vehicles in their approaches, which does not work without simulation. Real-world examples of autonomous driving with RL are almost non-existent with the notable exception of [17], which uses DDPG to demonstrate basic lane-following on 250 meters of empty road.

3.2 Learning by Cheating

Learning by Cheating (LBC) [7] is an imitation learning approach that was considered state-of-the-art on the previous CARLA benchmarks. LBC was released alongside Leaderboard as a baseline approach with an official open-source implementation, which our own method is based on. LBC is a two-stage approach whose key insight is the use of *privileged information* to train strong experts. LBC asserts that direct sensorimotor learning conflates two difficult learning tasks: perception and planning.



(a) Privileged agent - left to right flow (b) Sensorimotor agent - right to left flow

Figure 3.1: LBC overview, figure courtesy of the authors

To avoid this, LBC trains an intermediate expert that "cheats" by using privileged information that is not normally available to driving agents. By allowing the privileged agent access to extremely strong visual cues (a bird's-eye-view semantic segmentation of the agent and its surroundings), the perception task is essentially solved and the agent can focus on just learning how to act. The privileged expert is then able to provide strong supervision to the sensorimotor agent trained in the second stage which fully solves nearly all of the prior CARLA benchmarks.

The notion that developers should not be afraid to "cheat" during training and use information that would not normally be available at test time is an especially interesting aspect of this work that we draw attention and emphasis to. Simulation is *crucial* to this approach because it is impossible for real-world agents to have ground-truth, privileged knowledge about the environment. For example, there is no possible way to retrieve a BEV semantic segmentation image in the real world because such a sensor does not exist. The closest equivalent workflow using real-world driving data would be to hire human workers to manually annotate the positions, orientations, and states of other traffic participants and important cues like traffic lights, lane markings, stop signs, and crosswalks. This is clearly difficult to scale without enormous financial cost and time, and it is unclear how annotators would be able to determine distance, speed, and other information from raw sensor data. Human-labeled data are almost certain to contain errors that would not be present in simulation data annotated using privileged information.

3.2.1 Privileged Agent

The privileged agent trains a neural network that takes a bird's-eye-view (BEV) semantic segmentation image of the agent and its surrounding area as input, and predicts where it thinks an expert agent would be at various timesteps in the future. LBC uses an autopilot expert that navigates the environment using a set of hard-coded rules. Formally, given a dataset $\mathcal{D}$ comprised of BEV images M_t of the environment at time t and the ground truth future locations w_t of an expert over T future timesteps, the training objective for a privileged LBC model f parameterized by θ is:

$$\min_{\theta} E_{M_t, w_t \sim \mathcal{D}}[||w_t - f_\theta(M_t)||_1] \tag{3.1}$$

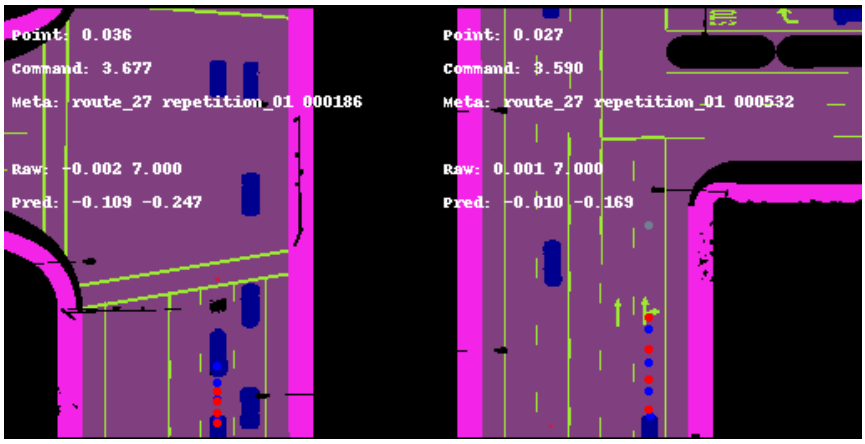


Figure 3.2: BEV images (M_t) of two samples. Agent is at the center-bottom of each image facing upwards. Red dots are LBC predictions $f_\theta(M_t)$, blue dots are ground truth w_t.

Since Leaderboard provides a high-level command at test time, LBC sets up a conditional policy that chooses the appropriate output branch depending on the given command c. Under the hood, the LBC privileged model sees the BEV image input M_t as a $(N_{\text{classes}}, H, W)$ tensor, where each (x, y) location has a N_{classes}-dimensional one hot vector describing what class the pixel belongs to. This is passed through a backbone network $b_\theta(M_t)$ which predicts a (T, H, W) tensor of logits. Each (H, W) slice of this tensor can be thought of as a probability heatmap predicting where the expert agent would be at time t. To retrieve the predicted points, we apply a spatial

softmax operation on $b_\theta(M_t)$ to recover an (x, y) coordinate for each of T heatmaps, i.e., $f_\theta(M_t) = \sigma(b_\theta(M_t))$. To produce a steering/throttle command, LBC averages the first two predicted points ($t = 0.5$s and $t = 1.0$s) to get a target waypoint, and computes a target speed from the distance between the two waypoints. These are passed to a PID controller which produces steering angle/throttle controls that we ask the simulator to execute.

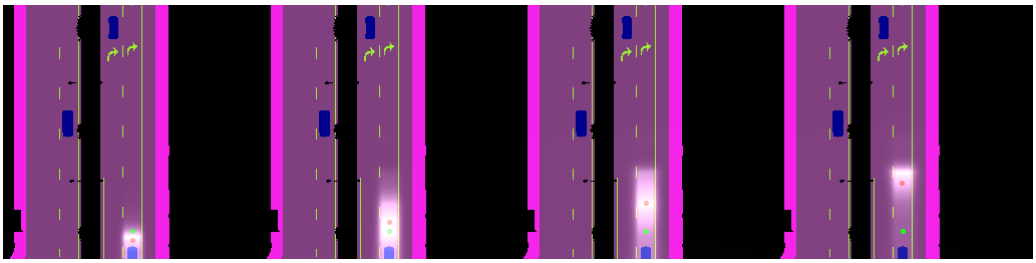


Figure 3.3: Visualizing each of the T heatmaps in $b_\theta(M_t)$ which has output shape (T, H, W). Each (H, W) heatmap is plotted in white over M_t.

3.2.2 Sensorimotor Agent

While the privileged agent cannot be submitted for Leaderboard evaluation, it can be used to provide strong expert supervision for the sensorimotor agent. They have the advantage of being highly scalable (i.e., not a human expert) and their predictions can provide dense ground-truth supervision to the sensorimotor agent at any M_t without requiring an actual expert rollout from that state. At its core, the sensorimotor agent operates exactly the same way as the privileged agent, with the only difference being the choice of input. Formally, given a fixed privileged expert $f^\dagger$, and an RGB image I_t in addition to M_t defined above, we express the training objective of a non-privileged sensorimotor agent g parameterized by ϕ as

$$\min_\phi E_{I_t, M_t \sim \mathcal{D}}[||f^\dagger(M_t) - T_p(g_\phi(I_t))||_1] \tag{3.2}$$

where T_p is a constant, known transformation from RGB camera frame (I) to BEV semantic segmentation frame (M). g_ϕ uses the same heatmap/softmax approach for retrieving points as f_θ. Importantly, $\mathcal{D}$ can be a dataset gathered by g while it is

actively exploring an interactive environment. Since f is a neural net, it can provide targets for g at any M_t without requiring an actual rollout (unlike data collection with the autopilot). Training both f and g requires access to privileged BEV images, but *testing* of the sensorimotor agent g is completely legal by Leaderboard standards - it just uses a standard RGB camera. This would also theoretically allow for the sensorimotor agent to be deployed in the real world, although the `sim2real` gap needs to be addressed.

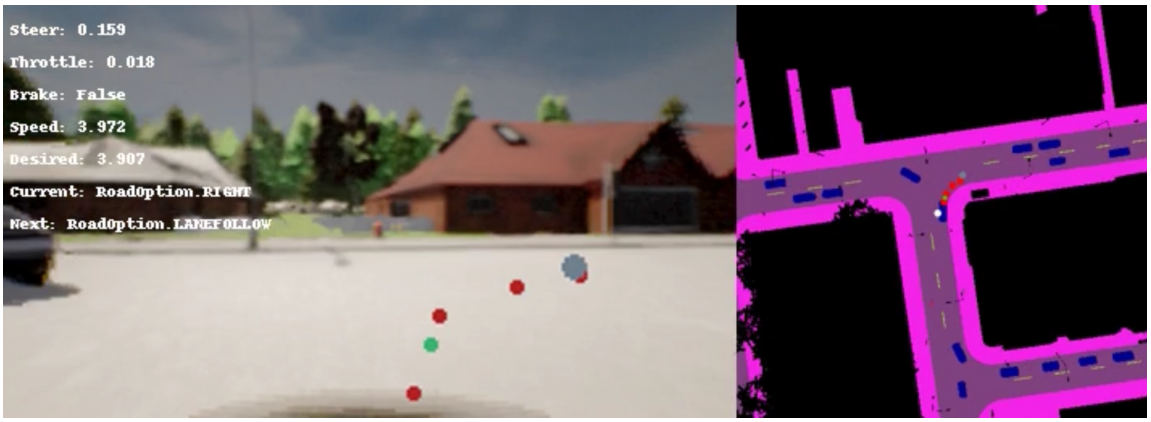


Figure 3.4: RHS: BEV semantic segmentation map with agent in the center and privileged prediction $f_\theta(M_t)$ in red dots. Dark gray dot is a coarse Leaderboard waypoint. LHS: RGB first-person camera view which is the input to g, and $f_\theta(M_t)$ transformed into image view. g aims to predict points that are close to the red points.

3.3 Failure Analysis

Comprehensive failure analysis is crucial for uncovering the weaknesses of autonomous driving methods and coming up with ways to address them. In this section, we'll perform an in-depth analysis of LBC's performance on Leaderboard that was made possible by our contributions to the Leaderboard evaluation toolkit. Three distinct agents play a role in the LBC training process: the autopilot, the privileged agent, and the sensorimotor agent. We run each agent for 10 repetitions on each validation route to establish a baseline. We will examine the most common failure modes of the image agent and attempt to diagnose why they might occur. We will also discuss common failure modes of the privileged agent and autopilot, and how they might

affect the downstream sensorimotor agent's performance. Note that no LBC agent is set up to handle stop sign infractions.

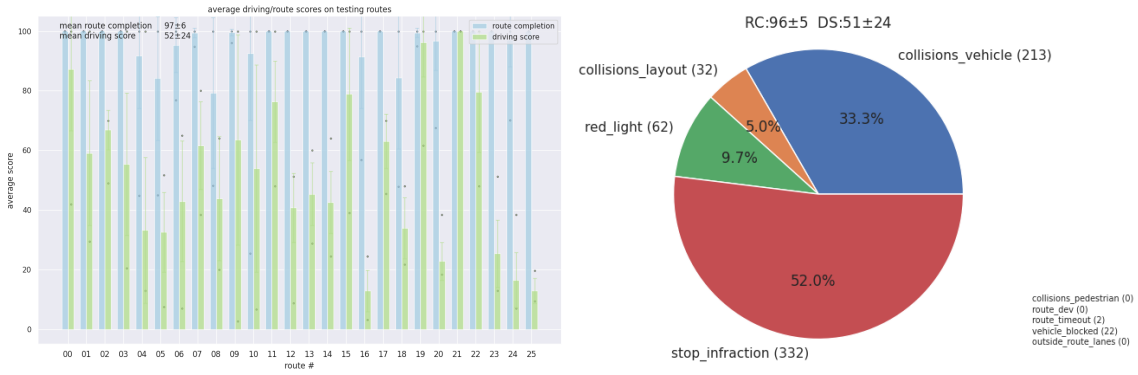


Figure 3.5: Average metrics for the autopilot

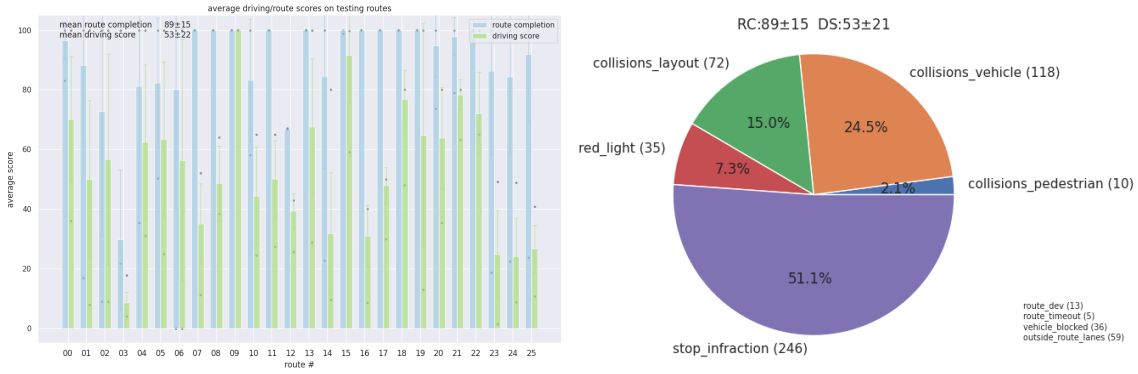


Figure 3.6: Average metrics for the privileged agent

3.3.1 Running red lights

From Figure 3.7, it's clear that red light infractions make up the plurality of infractions for the sensorimotor agent despite being a minority of infractions for the privileged agent and autopilot. Besides imposing additional penalties on the driving score, red light infractions are also a trigger for other infractions. When the ego agent runs a red light, it is very likely that the agent will also collide with another vehicle or follow it down the wrong turn (terminating the route). This is somewhat of a surprising result because the rules of traffic intersections are very simple - agents go on green,

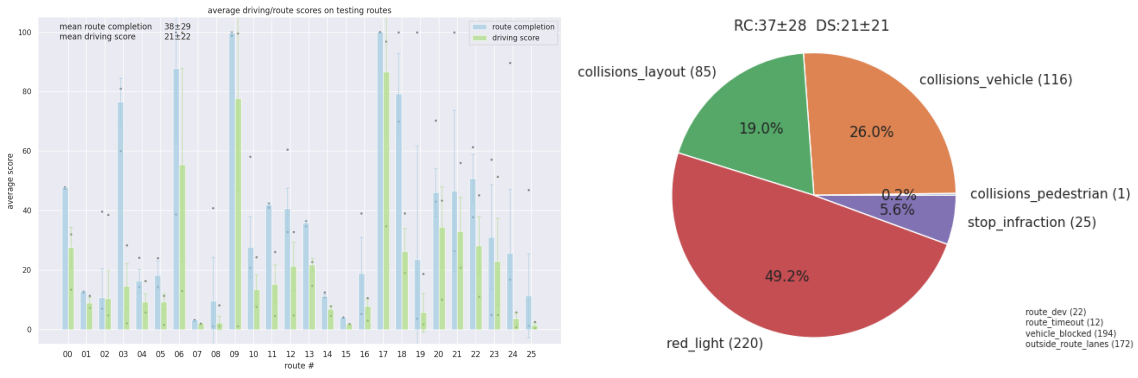


Figure 3.7: Average metrics for the sensorimotor agent

slow down/stop on yellow, and stop on red. Why then, does the sensorimotor agent suffer so much from red light infractions? It cannot be a downstream effect of the autopilot or privileged agent, because red light infractions make up a significantly lower proportion of their errors. We see three primary reasons for the sensorimotor agent's red light infractions.

Car-following

One well-known issue with neural networks is that they may not learn to use the desired cues from a given input. This is sometimes stated as a strength of deep learning - neural networks may learn to use better cues in their decision-making that humans do not use. In this case, however, the issue clearly manifests in an undesirable way - the sensorimotor agent is heavily influenced by how *other agents* are acting. In other words, the agent is prone to following the other agents it can see in its cameras. Specific to traffic light intersections, there are two cases where this occurs. If the agent is waiting behind a non-ego agent at a red traffic light, the car-following cue works well. Non-ego agents rarely run traffic lights, so following them into an intersection is generally a good idea. However, if the sensorimotor agent is directly in front of the traffic light, it will often follow other agents that are at the intersection that are not in its own lane. For example, when the traffic light for cross-traffic turns green, the sensorimotor agent will often follow those vehicles into the intersection even though its own light is red.

This specific issue is difficult to address because in most cases, car-following is

a good cue - the first situation described above is one example of this. So long as the ego agent follows the correct car, it is feasible to make it through an entire route with this cue. In practice, however, this is statistically almost impossible - there would have to be one (or multiple) agents that just so happen to guide the ego agent along the correct route. The alternative solution would be to completely or partially ignore the car-following cue in favor of other cues. It is not immediately clear how to accomplish this on existing datasets without significant effort. One approach would be to retroactively "drop-out" other agents from training data by somehow segmenting them out of images to simulate their absence during data collection. However, this would affect how the rest of the situation played out and introduces issues of causality. A simpler approach might be to create a separate dataset where there are no other agents at all, and ask the agent to learn on this dataset - perhaps in combination with datasets where other agents are present. This would remove the car-following cue since there are no other cars to follow, forcing the model to rely on other cues in the environment.

Small traffic lights

Clearly, the sensorimotor agent does not always act on the right cues at intersections. There are generally only two cues that the agent really needs to use - the state of the traffic light and the presence of any vehicles in the intersection. The traffic light is an especially strong cue and it is not hard to form a decision rule with respect to its state. We believe that one reason traffic light state is often ignored is because traffic lights are small. In most LBC camera images, traffic lights make up a very small part of the input images with the light color often being hard to distinguish from the rest of the traffic light. This makes it difficult for the traffic light state to play a significant role in decision-making - especially since other cars are (relatively) much larger, exacerbating the car-following issue. An empirical observation in support of this theory is that the sensorimotor agent appears to perform much better at small intersections where the traffic light is closer to the agent (sometimes only a couple feet in front and to the right of the agent). At large intersections, the ego agent will often halt and cause a blocked agent timeout, or wrongly follow cross-traffic.

There are a few ways to address this issue. The most common approach in

industry is to adopt a modular driving approach that specifically sets up a traffic light state identification task, and uses the output of that module to inform downstream decision-making. This is essentially an object detection/semantic segmentation task for which there exist many high-quality solutions. However, LBC is an end-to-end method which makes this solution difficult to quickly integrate. A different approach is to introduce an auxiliary task that helps the LBC model learn more traffic light relevant features. Concretely, we can set up a semantic segmentation reconstruction task that heavily weights traffic light reconstructions.

This approach is used in the top 2 best-performing Leaderboard approaches with both explicitly citing it as a way to significantly improve generalization. MaRLn introduces both a semantic segmentation task and a traffic light state prediction task as part of their bevy of auxiliary "implicit affordance" tasks. Using both of these tasks increases the percentage of successful traffic light intersection traversals from 0% to 96.5% on the CoRL2017 benchmark. The WOR authors performed a similar ablation study and showed that introducing a general semantic segmentation task increased test performance on NoCrash from 46% to 76%. One more way this can be addressed is with the introduction of an additional camera sensor that is specifically meant to capture distant objects. We can create a telephoto lens by configuring the field of view and height/width of a camera to zoom in on the region directly in front of the agent. This would help increase the relative size of any traffic lights in the input image and hopefully make a stronger cue for models to use.

Short yellow lights

A significant number of red light infractions can be attributed to the following observation: traffic lights in Leaderboard switch from yellow to red extremely quickly. Evaluation videos show that this is the primary reason for the majority of red light infractions caused by the privileged agent and autopilot expert. Short yellow light times cause a very common failure mode where the ego agent will approach an intersection with a traffic light that is signaling green. As the agent begins to enter the intersection, the traffic light will switch to yellow, then almost immediately switch to red. Since the ego agent is usually still in the intersection at this point, it incurs a red light infraction. Additionally, cross-traffic will usually immediately move into

the intersection when the ego agent's light turns red, setting up highly dangerous situations that can often end with a collision or a route deviation.

One immediate solution could be to extend the time that traffic lights stay in yellow before transitioning to red when collecting training data and/or performing local evaluations. However, this poses a major problem: the held-out test test on the public evaluation server uses the default Leaderboard parameters, so short yellow light times will still occur. We believe that this is a *major weakness* of Leaderboard's realism. Traffic lights in the real world rarely have such a short transition time. Furthermore, even when the ego agent's light turns red and cross-traffic gets the go signal, they will rarely enter the intersection as immediately. [34] suggests that human drivers have a red-to-green light response time of 0.75 to 1 second. In contrast, when Leaderboard agents see a green light, they will immediately proceed at high speed into the intersection with no delay.

3.3.2 Collisions

Autopilot

It is evident from Figure 3.3 that the autopilot expert is prone to collision. In fact, it receives the most vehicle collision infractions of all three agents. This could be partially explained by its higher average route completion, but qualitative analysis of the autopilot's evaluation videos show that its rules-based collision avoidance logic is not robust against certain edge cases. There are two notable situations where the autopilot almost always collides with vehicles and pedestrians. Occasionally, non-ego vehicles will run a red light and enter the intersection when the autopilot is also entering the intersection. Whether this is intentional or buggy Leaderboard behavior, the autopilot logic does not recognize the presence of these vehicles in its path and will run full speed into them. The other situation occurs when Leaderboard spawns a pedestrian or biker that runs across the road directly in front of the agent, which does not recognize the danger and will also run full speed into them. Autopilot planning logic should be reexamined and tuned to avoid failure on these edge cases.

Privileged agent

Although autopilot demonstrates very consistent vehicle collision failures, that does not appear to be reflected in the downstream privileged agent, as shown in Figure 3.6. The privileged agent exhibits much more robust collision avoidance and will slow down for crossing pedestrians or other agents in intersections. In fact, some of its collision infractions are not due to incorrect behavior at all. In route 20 for example, a biker crosses the road at one point that the privileged agent correctly slows down for. The biker continues into oncoming traffic and always gets hit, and is subsequently launched into the privileged agent's vehicle which incurs an infraction through no fault of its own.

However, there are three failure modes that make up the majority of privileged agent vehicle collisions. First, it will often hit cross-traffic when crossing stop intersections. This is expected since LBC does not handle stop signs at all and will always run them. Second, the privileged agent will often side-swipe vehicles when changing lanes on the highway - this is caused by the BEV semantic segmentation input only showing areas in front of the agent. One simple solution is to include the area behind the agent as part of the semantic segmentation input, or alternatively include a LiDAR sensor for detecting objects all around the agent. Finally, the privileged agent will sometimes drift into the opposite lane and hit oncoming traffic. This typically happens when the road curves right or left, luring the agent into a neighboring lane. This failure mode is quantitatively measured by the 59 outside route lanes infractions recorded during evaluation. Overall, however, the privileged agent is very adept at avoiding vehicle collisions and provides strong supervision for the sensorimotor agent in this regard.

Static collisions also make up a significant amount of privileged agent infractions. While vehicle collisions typically emerge through sequences of events that play out uniquely in each run, static collisions occur much more consistently for particular routes. In route 1, for example, the agent proceeds along a road that suddenly curves sharply towards the right. The privileged model overcompensates the right turn and will always run into the railing on the side of the road, guaranteeing at least one static collision on this route. On route 10, the privileged model is asked to drive straight along a highway in the rightmost lane. However, it will often be led astray into a

highway exit only to correct itself at the last minute, colliding with the safety barrels that are placed at exits. The agent will also frequently crash into barriers later on in the route when it must make a sudden right turn to exit the highway. For the most part, static collisions occur because of the agent's inability to keep in the correct lane, which is a common cause of failure between this agent and the sensorimotor agent.

Sensorimotor agent

The sensorimotor agent struggles with frequent collisions on Leaderboard, which is a surprising result considering its strong performance on NoCrash. Despite achieving only about 40% of the privileged agent's average route completion score, it incurs the same number of vehicle collisions and actually causes more static collisions. Interestingly, many collisions seem to occur for no immediately obvious reason, with behavior not always being consistent between two repetitions of the same situation.

Figure 3.8: Left: SM agent collides and continues to step on gas. Right: In different repetition, SM agent safely brakes behind car

We consider a case in route 3 (Figure 3.8) where the agent proceeds straight with a vehicle in front of it. Despite being in the same situation over ten repetitions, the ego agent runs straight into the back of the vehicle with no hint of slowing down in three repetitions, while demonstrating normal collision avoidance behavior in the other seven. There is no major visual difference between all repetitions with the exception of the model of cars in oncoming traffic. One possible explanation is that the model might be overfitting - if this street shows up in the training data with no other cars on the road, the model may be inclined to predict fast forward driving. However, with other vehicles present at test time, the learned behavior for this specific road

may conflict with usual collision avoidance behavior.

The agent also displays some worrying behavior in this case - even after collision with the vehicle, it continues to execute forward motion. Besides being incredibly dangerous in the real world, this is confusing behavior - why would the model expect an expert to continue driving forward when it has collided with another vehicle? This may be due to a distribution shift at test time. The sensorimotor agent will inevitably make mistakes and deviate from good driving behavior into dangerous situations. Since the autopilot and privileged agent are both very good at avoiding these situations, they are not well-represented in the training data. This leads to unpredictable behavior from the sensorimotor agent since it has rarely seen good examples of what to do in or near collisions.

Overall, the failure modes for sensorimotor agent collisions are diverse and difficult to categorize. A few examples are: running into a biker crossing the street without any hint of slowing down, side-swiping passing cars when changing lanes, taking a turn too wide and running into oncoming traffic on the opposite lane, taking a turn too sharply and running into road barriers. This points to one fundamental issue with imitation learning - there is no way to directly penalize bad behavior and incentivize good behavior, suggesting that cost-sensitive learning (i.e., reinforcement learning) may be a suitable alternative.

3.3.3 Lane-keeping issues

While significant route deviations will terminate route deployments, agents are not explicitly penalized for slightly deviating from the lane they are supposed to be on. However, we observe that the inability to keep in the correct route lane puts agents at much higher risk for both vehicle/static collisions and blocked agent timeouts. This issue is mostly negligible for the privileged agent and autopilot, but it is a significant issue for the sensorimotor agent. The privileged agent received 59 outside route lane infractions that are mostly minor deviations (e.g. strays onto a neighboring lane). On the other hand, the sensorimotor agent racked up a whopping 172 infractions out of 260 total runs with many serious deviations.

There are three primary lane-keeping failure modes, examples of which are shown in Figure 3.9. The majority are caused at intersections, where the sensorimotor agent

Figure 3.9: Top and bottom right: agent makes intersection turn too narrow and wide. Top left: agent chases the route waypoint off-road. Bottom left: agent mistakes highway shoulder for lane

will make turns imprecisely. On the top right, the agent makes too sharp of a left turn and gets blocked in front of the highway barriers. On the bottom right, the agent struggles to decide between going straight and turning right. Eventually, it correctly chooses to go right but has made too wide of a turn and either hits the purple vehicle or gets blocked by it. Besides intersections, a common failure mode on highways is when the agent mistakes the shoulder as another lane to change into as shown on the bottom left. The agent is asked to change to the right lane, which it executes successfully, but it then decides to change lanes again and ends up hitting the highway railing to the right of the shoulder. Finally, we observe that the coarse Leaderboard route waypoints significantly influence the predicted sensorimotor agent waypoints. The top right image from route 0 shows the model predicting a hard turn off the road across sidewalk and grass in an attempt to reach the purple route waypoint. This generally ends in a vehicle collision once it tries to re-enter the road.

Clearly, lane following is a strong driving cue that often leads to infractions when not followed. With imitation learning, it's not immediately clear how to explicitly

enforce this. One approach might be to include auxiliary tasks for lane line predictions, but lane lines are small and difficult to see in camera images. Also, there are no lane lines in the middle of intersections. On the other hand, we could once again turn to reinforcement learning to explicitly reward the agent for staying in the correct lane to encourage the correct behavior.

3.4 Technical Approach

We scope our project to focus exclusively on developing a stronger *privileged* agent using a hybrid imitation/reinforcement learning approach. Online reinforcement learning is notoriously sample-inefficient and time-consuming to run, so we experiment with offline reinforcement learning, hence the name "Offline Imitative Reinforcement Learning" (OIRL). Using OIRL to successfully train a privileged agent stronger than LBC's would theoretically make it possible to immediately train a stronger sensorimotor agent. This could be done by simply replacing LBC's second-stage expert with the stronger OIRL privileged agent. Alternatively, it should also be possible to directly train the sensorimotor agent using OIRL with any choice of privileged expert. In the following sections, we will review the method we base our approach on and describe experimental modifications to the training scheme.

3.4.1 Deep Q Learning from Demonstrations

Expert imitation is generally a strong cue for good driving, but one significant weakness is that there is no straightforward way to explicitly penalize student actions that lead to bad outcomes. For example, the LBC learning objective is solely concerned with expert imitation - there is no consideration of what the actual outcome of an action is. On the other hand, reinforcement learning offers a way to directly penalize dangerous actions via reward shaping. However, end-to-end RL for driving has typically struggled to achieve good performance beyond relatively simple maneuvers like cruise-control and merging. It is difficult to manually construct a reward function that describes good driving behavior because there is no concrete definition of "good" driving that applies to every situation. On the other hand, bad behavior is much simpler to define - agents must avoid collisions, traffic infractions, etc. Given this

observation, one approach is to retain expert imitation as a strong driving cue while introducing reinforcement learning as a way to "fine-tune" dangerous behavior.

This intuition inspires a method called "Deep Q Learning from Demonstrations" (DQfD) which we heavily base our approach on. DQfD is a hybrid imitation learning/reinforcement learning method whose main innovation is using a combination of TD loss and supervised imitation loss to train a Q network. Formally, their loss function is

$$J(Q) = J_{DQ}(Q) + \lambda_1 J_n(Q) + \lambda_2 J_E(Q) + \lambda_3 J_{L2}(Q) \tag{3.3}$$

$$J_E(Q) = \max_a [Q(s,a) + l(a_E, a)] - Q(s, a_E) \tag{3.4}$$

$$J_{DQ}(Q) = ||Q(s_t, a_t) - (r_{t+1} + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}))||_2^2 \tag{3.5}$$

$$J_n(Q) = ||Q(s_t, a_t) - R_t^{(n)}||_2^2 \tag{3.6}$$

where J_{DQ} and J_n are respectively TD(0) and TD(n) losses and $R_t^{(n)}$ is the n-step return defined as

$$R_t^{(n)} = r_{t+1} + \gamma r_{t+2} + ... + \gamma^{n-1} r_{t+n} + \gamma^n \max_{a_{t+n}} Q(s_{t+n}, a_{t+n})$$

J_E is an expert imitation loss implemented with a margin function $l(a_E, a)$ that is 0 when the action a chosen is the expert action a_E and positive otherwise. This pushes the Q values of expert actions to be higher than other actions. J_{L2} is an L2 regularization loss on the network weights for generalization.

In this approach, the authors assume access to an RL environment for training and a dataset of expert demonstrations in this RL environment. They do not assume access to the expert itself. Training proceeds in two stages: offline and online. The offline stage is trained only on the expert demonstrations, and the entire loss function $J(Q)$ is applied. The online training stage proceeds in a DQN-like manner, with *two* separate replay buffers being used. $\mathcal{D}_{\text{expert}}$ contains the fixed dataset of expert demonstrations while $\mathcal{D}_{\text{env}}$ contains a dynamic experience buffer collected with the behavior policy (derived from the DQN being trained). Importantly, since DQfD does not have the expert at training time, samples from $\mathcal{D}_{\text{env}}$ cannot have the expert loss J_E applied. A prioritized replay mechanism samples from both $\mathcal{D}_{\text{expert}}$ and $\mathcal{D}_{\text{env}}$

to favor high-loss training samples. DQfD's experimental evaluations on a bevy of Atari games show two advantages of this approach: increased sample efficiency (i.e., more rapid initial performance gains), and asymptotic performance on par with or occasionally surpassing state-of-the-art methods.

3.4.2 OIRL

Let's consider how we can configure the privileged LBC model f_θ as a Q_θ network that uses the same state/action representation. As we specified in Section 3.2.1, f_θ consumes a $(N_{\text{classes}}, H, W)$ map tensor M by passing it through a backbone network b_θ and producing a (T, H, W) tensor of logits (Figure 3.3). In the typical LBC approach, f_θ takes a softmax expectation over the heatmap logits to retrieve T points as the predicted action. Instead, Q_θ views these logits as T slices - one for each future timestep - of value heatmaps of size (H, W). We define an action p as the T points chosen over the heatmaps, with the Q value being the sum of the value heatmaps $b_\theta(M)$ indexed over the T points. Since the action space is discrete, we can compute a policy from this Q network as the union of (x, y) coordinates of maximum value over each of T heatmaps:

$$Q_\theta(s = M, a = p) = \sum_{p_t \in p} b_\theta(M)|_{p_t} \tag{3.7}$$

$$\pi_\theta(a = p | s = M) = \bigcup_{t}^{T} \left[\underset{x,y}{\text{argmax}}\ b_\theta(M)|_t \right] \tag{3.8}$$

Now that we can formulate Q_θ using the same state/action representation as LBC's f_θ, we can train the network with OIRL. We use a fixed privileged LBC expert $f^\dagger$ to collect a dataset $(s_t, a_t, r_t, s_{t+1}, d_t) \sim \mathcal{D}$ by running the privileged agent on all 50 Leaderboard training routes for 4 repetitions each. $\mathcal{D}$ contains about 1.3 million samples collected at 20Hz, which is approximately 19 hours of driving data. Then, we instantiate learning agent Q_θ and train it following Equation 3.9. This loss resembles the DQfD loss, but removes the TD(0) loss and network weight regularizer. We use TD(n) loss for $J_n(Q)$ where $n = 20$ (1-second lookahead) and L1 point loss for $J_E(Q)$ similar to Equation (3.1). There are two milestones we would like to achieve: (1) can we get the π_θ policy yielded by Q_θ to match the performance of $f^\dagger$? and (2) can π_θ

actually surpass $f^\dagger$ with entirely offline RL training?

$$J(Q) = J_n(Q) + \lambda_E J_E(Q) \tag{3.9}$$

Note that while our approach is entirely offline, it is straightforward to transition to online learning if desired. One key difference (and possible novelty) of this approach from DQfD is that we would have access to $f^\dagger$ at training time, so the replay buffer would not have to be split between fixed expert demonstrations and novel demonstrations from our student. Concretely, we could apply Equation 3.4 to novel states encountered by π_θ during exploration. This resembles a typical DAgger setup, with the key difference being the combination of TD losses with imitation losses. Having the expert available during online training may lead to more rapid improvement and possibly better asymptotic performance compared to DQfD.

3.4.3 CARLA as an RL Environment

RL environments must yield a reward scalar in response to actions that agents take. Developers often modify the reward structure of RL environments in an attempt to evoke specific behaviors, which is referred to as reward engineering. However, CARLA cannot be used as an RL environment straight out of the box. As part of this approach, we modified the Leaderboard toolkit to yield RL rewards in response to agent actions at runtime.

Our reward structure is composed of two components. Intuitively, we would like to encourage the agent to do two things: progress along the route and avoid incurring infractions. Under the hood, Leaderboard only produces route completion score at the end of a run, which is used to compute the final driving score. However, we need to access route completion scores *during* data collection so we can use it as part of the reward. We introduce additional bookkeeping to the Leaderboard toolkit which allows us to directly use the change in route completion as part of the reward. However, route completion tracking is not continuous - it is possible for the ego agent to progress along the route, yet not register any progress in route completion. At a given tick, the change in route completion (which is directly used as the route reward component) is either 0 or between 0.01 and 0.5. This may provide an inconsistent learning signal to the RL agent but we can address this using n-step learning, which

smooths out the discrete route completion reward.

Leaderboard records infractions in a similar way to route completion - all infractions are written to the log file that is accessed at the end of the run. We again introduce additional bookkeeping to allow for infractions to be detected at runtime. However, it is not as straightforward to determine the specific rewards that each infraction type should yield, since they apply a multiplier rather than a flat penalty. We transform the multiplier into a flat penalty by simply multiplying it by -100 e.g. a pedestrian collision normally applies a 0.5x driving score penalty, so it yields a -50 reward during RL data collection.

3.4.4 Prioritized Experience Replay

In "vanilla" deep reinforcement learning approaches, experience tuples are uniformly sampled from the replay buffer during training. Prioritized experience replay (PER) takes a different approach by observing that not all samples are equal - some are more instructive for learning good behavior than others. There are a few variations on how this is implemented, but the general approach is to sample experience tuples proportional to the magnitude of their TD loss. However, this is difficult to use with our dataset - even if an experience tuple has significantly higher TD loss, its sampling probability will see a small, negligible increase given the 1.3M total samples present.

We adopt a modified prioritized replay mechanism inspired by our earlier motivation for OIRL. The privileged expert drives well in most cases, but we are trying to use cost-sensitive TD learning to fine-tune its behavior in specific cases where it causes Leaderboard infractions. We augment the LBC data collection code to also record if an infraction occurred as a result of the data agent's action. Our dataset $\mathcal{D}$ contains 79 pedestrian collisions, 173 static collisions, 357 vehicle collisions, and 22 traffic light infractions. While training, we create an additional replay buffer only made up of infraction samples. We define an infraction sample as any sample where the agent caused an infraction in the next n timesteps, where n is the hyperparameter used for TD(n) loss. We introduce a hyperparameter κ that defines the proportion of each training batch that must be infraction samples. During training, the proportion starts at 0 and is linearly annealed to κ over the first half of training, and remains fixed for the second half.

3.4.5 Expert Imitation Masking

A problem emerges when training OIRL with prioritized experience replay and no additional modifications. Given an infraction sample, the TD loss is supposed to encourage the OIRL model to predict a low value for the given state and predicted points and instead choose other points. However, the expert imitation loss gives the complete opposite signal and strongly encourages the model to pick the exact same points. Rather than assigning a low value to just the expert's points, the model will generally predict lower values for *every* point in the map. To address this, we *mask* the expert imitation loss by zeroing it out and only use TD loss for infraction samples.

3.5 Experiments

3.5.1 Setup

We train each model on the dataset for 50 epochs and pick the best model on validation loss to benchmark. All experiments use $\kappa = 0.5$ and $T = 20$ (if appropriate) and each model is initialized using the pretrained privileged LBC model weights. The best-performing model on validation loss is picked as the model used for benchmarking. We run each model on the Leaderboard validation routes for 4 repetitions each. We record the standard Leaderboard metrics and generate our custom visualizations and plots. LBC uses an early version of Leaderboard that includes 3 of the 10 NHTSA pre-crash scenarios. The "ControlLoss" scenario briefly overrides the agent's actions with random noise and models situations like hydroplaning or black ice. The other two scenarios are variations on a pedestrian or biker illegally crossing the street. We refer to the privileged model as the "baseline" in all discussion, and its results are shown in Figure 3.6. Note that the baseline was run for 10 repetitions on each validation route as opposed to 4 repetitions for these results - this is reflected in the larger numbers of infractions on the baseline's infraction pie chart.

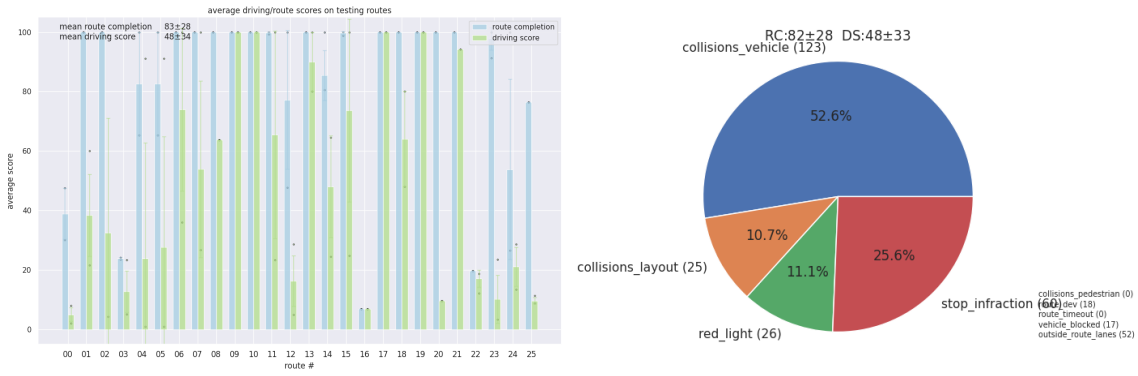


Figure 3.10: Margin-only average metrics

3.5.2 Results

Margin

Intuitively, retraining the LBC model as a deep Q network using only max margin expert imitation loss should yield a model with similar performance to the imitated expert. While this agent has similar numbers on the validation route compared to the baseline, we do see a decrease in both route completion and driving score for this model. Why might this be the case? One hypothesis is that there might be some underlying advantage in training the model to predict waypoints via softmax heatmap extraction rather than argmax heatmap extraction. The expert margin loss pushes the values of pixels not immediately surrounding the expert's waypoint to zero, which may cause the gradient to disappear in non-expert regions. On the other hand, the spatial softmax should in theory allow loss gradients to backpropagate through a larger portion of the heatmap. This is evident when comparing the output layer activations (before softmax or argmax) of both approaches - the softmax approach's activations are more diffuse and spread out over the topdown view.

Additionally, the infraction pie chart shows a significant reduction of stop infractions compared to the baseline. However, this is misleading for a slightly complex reason. Due to the subtle differences in how both models predict waypoints for the same route, specific situations will consistently emerge - unique to each model. We consider an example from route 21 (Figure 3.12), where the agent must cross a stop intersection. Normally, both models would blow past this intersection and generate a

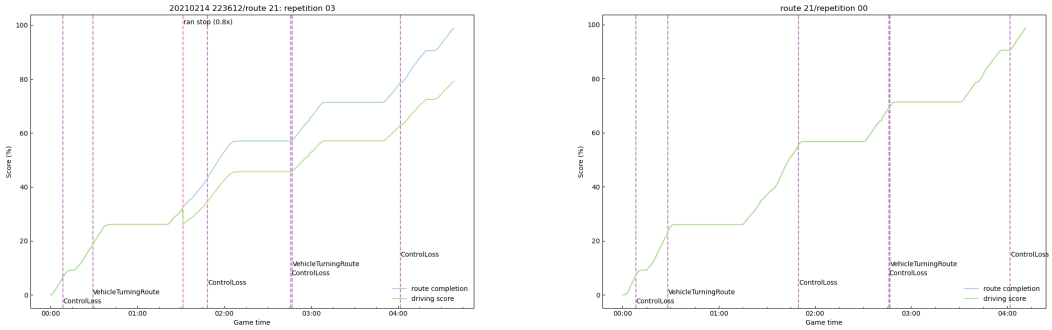


Figure 3.11: Left: baseline causes stop infraction. Right: margin agent avoids stop infraction

Figure 3.12: Left: baseline is in the correct lane. Right: margin agent is in the wrong lane, but is able to avoid stop infraction

stop infraction, but the proposed model does not seem to do this. This is surprising, since stop signs are not explicitly handled at all by either approach. By investigating the evaluation videos, we see a consistent sequence of events play out that explains this phenomenon. Prior to this intersection, the agent must make a right turn and stick to the right lane before arriving at the stop sign. The baseline agent correctly makes the turn but does not stop for the intersection, causing the infraction. On the other hand, the DQN margin agent always makes this turn wide and drifts into the left lane, coming to a stop behind the police cruiser (or occasionally hitting it). The agent proceeds into the intersection when the police cruiser does, avoiding the stop infraction despite not being explicitly programmed to handle it.

This is intriguing because the DQN agent's bad behavior (swinging wide on the turn, keeping to the wrong lane) actually masked the stop infraction's effect on the

final driving score. This also points to a weakness in the Leaderboard evaluation - the DQN agent should have been penalized for the stop infraction even though it was following the police cruiser's lead. The DQN agent never actually stopped in front of the stop sign, but it must have been close enough while stopped behind the police cruiser to count as correct behavior. This hints at more fundamental issues with the Leaderboard benchmark that we discuss in Section 5.1. Overall, however, the DQN margin approach achieves mostly comparable performance to the baseline and serves as a sanity check for a correct implementation of expert margin loss.

Margin, TD Loss

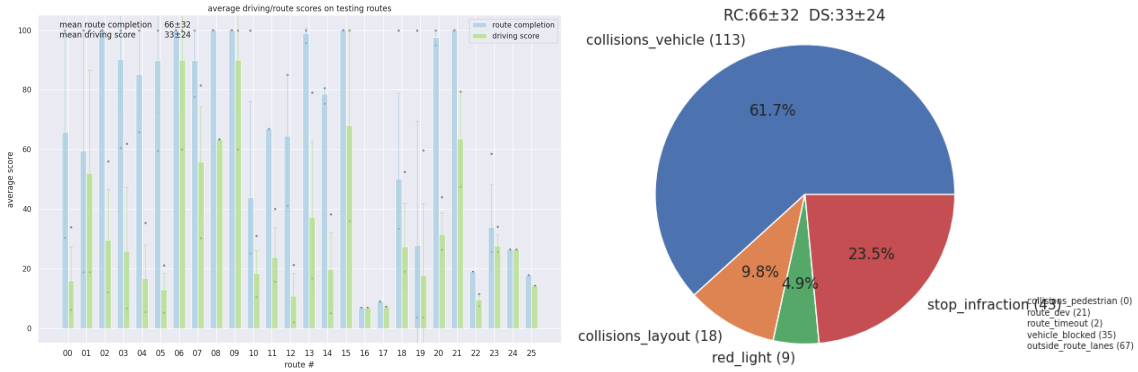


Figure 3.13: Margin and TD loss average metrics

Next, we investigate the results of applying TD(20) n-step loss alongside the expert margin loss. Ideally, the TD loss should improve upon pure imitation learning by fine-tuning the model in situations where the expert's action led to a bad outcome. This is possible because of the work we did to turn CARLA into an RL environment, complete with a reward structure we can use to specify desired behavior. However, Figure 3.13 shows that the introduction of TD loss decreases both driving score and route completion metrics on the validation set. This is an undesirable result and runs counter to cursory intuition. Why does performance degrade with the introduction of TD loss?

A clearer picture emerges when we look at visualizations of the model's output layer activations (Figure 3.14). The margin-only model shows a relatively clean activation map that is lit up in reasonable areas immediately surrounding the ego agent, while

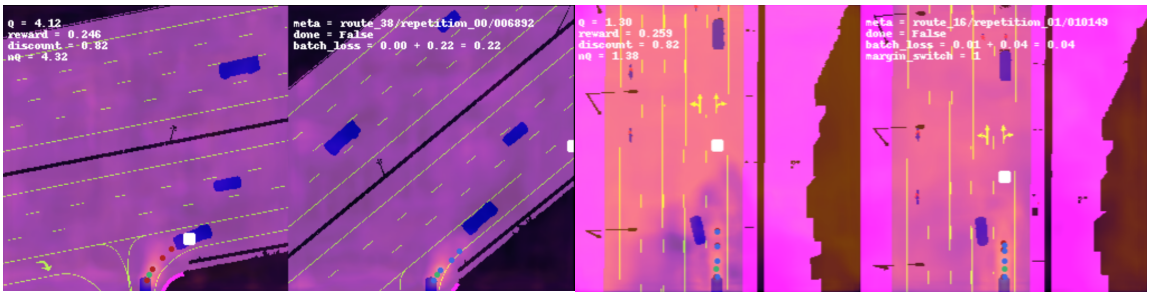


Figure 3.14: Left pair: margin-only activations. Right pair: margin and TD loss activations

the margin and TD model has significantly noisier activations on the output layer. This is an example of the *extrapolation error* phenomenon in reinforcement learning. When training, the DQN model may assign a high value to state and action pairs it believes to be good. However, since we are training entirely offline on a static dataset there is no opportunity for the agent to actually try out the suggested actions. Despite the apparent intractability and weakness of training reinforcement learning in an offline setting, we can observe that the margin and TD model still predicts reasonable actions to take, as the expert imitation cue is strong. While TD learning seems to cause extrapolation error, it may be useful if applied in the right situations - specifically infraction scenarios. This motivates our experiments with prioritized replay.

Margin, TD Loss, PER

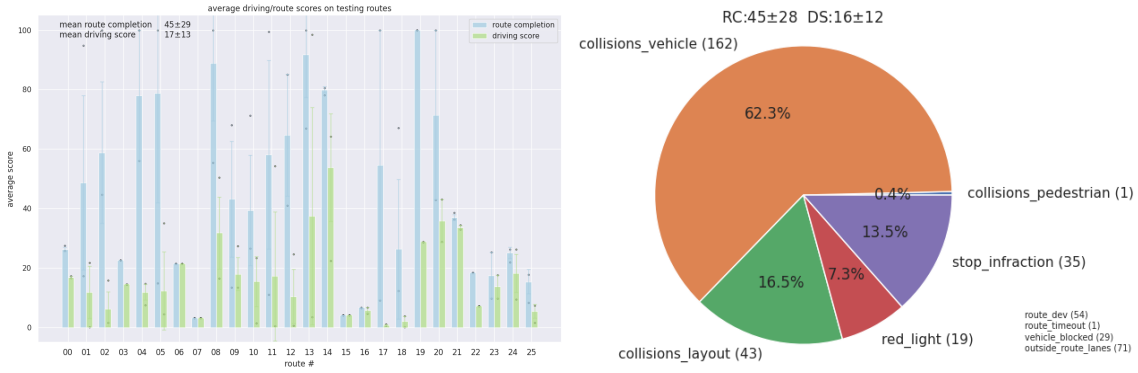


Figure 3.15: Margin, TD loss, PER average metrics

We will briefly describe the results of training with only prioritized replay and expert imitation without expert masking for infraction samples. As expected, the performance is significantly worse than the baseline. Since the expert model was also used as the data collection model, following the expert imitation cue at infraction samples reinforces the same behavior that led to the infractions found in the dataset. Despite achieving only 50% of the baseline's average route completion score, this model exhibits about 50% more vehicle collisions. The incidence of other infractions is also relatively high considering this model's low route completion scores.

Margin, TD Loss, PER, Expert Masking

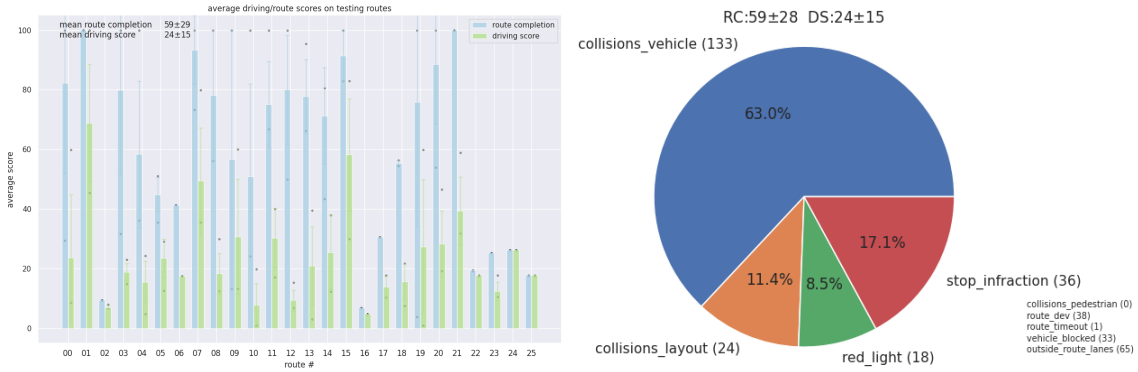


Figure 3.16: Margin, TD Loss, PER, expert masking results

As expected, reintroducing expert masking at infractions significantly improves the privileged agent's performance, but still does not come close to beating the baseline model. Comparing this model's average driving score and route completion scores (Figure 3.16) against the baseline's scores, we observe an across the board reduction in metric performance. While this model does show marked reductions in collisions and (relatively) fewer red light infractions compared to the previous models, this is likely due to the same phenomenon we described with the margin-only DQN agent. We also observe that route deviations are a significant weakness of this method. Despite only running each validation route for 4 repetitions as opposed to the baseline's 10 repetitions, this agent produced almost three times more route deviations.

What causes the agent to deviate from the route? One minor and relatively rare failure mode is when the agent needs to take a highway exit, but misses it and

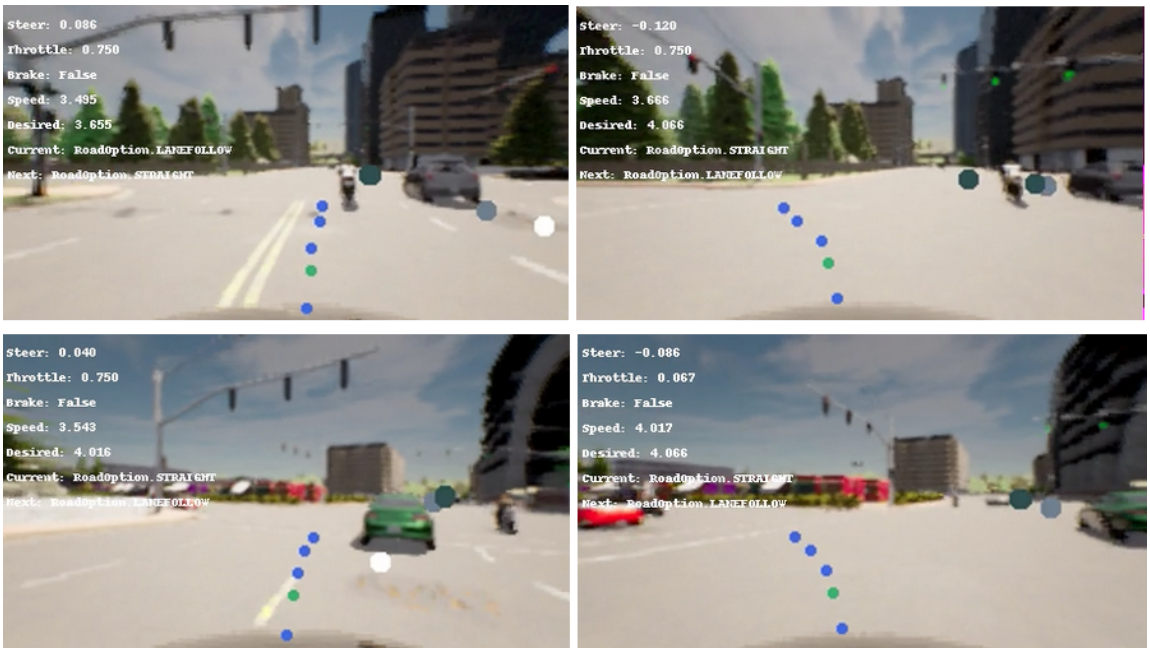


Figure 3.17: Top: In left image, agent drifts left and deviates from going straight in the right image. Bottom: Similar sequence of events that occur on a different intersection

continues straight. However, the primary cause of route deviations is that the agent struggles with staying in the correct lane, which is quantitatively supported by the "outside route lanes" counts in the infraction chart. Given 26 routes and 4 repetitions per route, the agent strays from the given route lane at some point on 65 of the total 104 runs. Qualitatively, the agent is very easily led into wrong lanes for a variety of reasons - it may see another car in the lane and decide to follow it, it might make too wide of a turn and align with the wrong lane, or it might simply drift to one side of the correct lane, and decide to suddenly jump over to the next one.

We can see specific examples of this behavior in routes 23, 24, and 25, which all end in route deviation terminations for the proposed agent. We observe a sequence of events common to all of these routes that leads the agent astray. In each case, the agent must cross an intersection and continue straight. However, in each case the agent is either already in the wrong lane, or is not well-aligned with its current lane. As the agent enters the intersection, it continues to drift more to the left, which sets off a cascading effect - the more to the left the agent is in the intersection, the

more likely it will think it should turn left despite the high-level command indicating otherwise. This may be due to a number of reasons, but the agent's poor lane-keeping ability clearly contributes to tricking it into making the wrong turn. Another reason might be due to the prioritized replay mechanism - intersections are common areas for infractions to happen, and the model may be overfitting to intersections that are overrepresented in the infraction dataset.

3.6 Discussion

We were unable to surpass the performance of the privileged expert with our approach and training modifications. Overall, we attribute this to two factors. First, offline reinforcement learning is difficult primarily due to *extrapolation error* and the inability to interactively try new actions in the simulator. As suggested at the end of Section 3.4.2, online training may be one way to address this particular issue. While this heavily resembles DQfD, there's one key difference: we have access to the privileged LBC expert during online training, which may help with sample complexity. Second, our parameterization of action space is very large. Given N pixels in a map there are N^T possible actions, the majority of which are never used. LBC contains an experimental module that consumes the predicted expert waypoints and directly predicts steering angle, throttle, and brake. One approach could be to discretize the action space into buckets, and train deep Q networks on this much smaller action space. In general, we believe the fundamental insight still holds true - pure imitation learning cannot teach agents to recognize the consequences of their actions, which can lead to very dangerous behavior. In the next chapter, we work with the most recent top-performer on Leaderboard, which completely eschews imitation learning in favor of offline reinforcement learning.

Chapter 4

Slightly Online Reinforcement Learning

4.1 Background

LBC demonstrated essentially perfect performance on every single CoRL2017 benchmark setup, and perfect or at least very strong performance on the majority of NoCrash setups. However, LBC performs significantly worse on Leaderboard and only achieves a driving score of 8.9% out of 100. In the year and a half that Leaderboard has been public, a bevy of different methods that make use of relatively new techniques (like attention mechanisms) to achieve better results have surpassed LBC. Despite being previously relegated to simple driving subtasks, reinforcement learning has surfaced in the two best performing approaches on Leaderboard.

At time of writing, the top-performing approach on Leaderboard is World on Rails (WOR), which our approach extends upon. Reinforcement learning theory is well-established, and the field has produced impressive results in the last decade with algorithms like AlphaGo and MuZero [27, 30]. However, WOR is notable for being part of a recent wave of offline reinforcement learning approaches that has become popular in robotics research over the past year . The WOR paper, for instance, was released a little over three months before time of writing. We will first cover the basics of offline reinforcement learning and prior work, and then describe how WOR

addresses the usually debilitating issues with offline RL.

4.1.1 Offline Reinforcement Learning

Offline reinforcement learning approaches are a somewhat surprising class of approach because they are fundamentally antithetical to the standard approach for RL. Generally, agents that are trained via RL have the opportunity to try out actions in the environment and receive scalar reward feedback. The reward is used to update the agent's model and help it figure out what a better course of action might be. In other words, RL agents are fundamentally allowed to interact with the training environment. However, in offline RL, the student model is trained with a static batch of data that it cannot add to (offline RL is often also called "batch RL"). While learning, the model has no way to interact with the environment and receive feedback for other actions it thinks might be good. This leads to what is known as the extrapolation error phenomena, where states or state/action pairs outside of the dataset often have arbitrary/inaccurate values. We point the reader to [20] for a comprehensive tutorial and review of offline RL.

Most "older" offline RL work focuses on robotic grasping tasks, which naturally meshes well with offline RL because of the need to generalize to a wide range of objects [19] [25]. These methods use approximate dynamic programming as well as more domain-specific algorithms to train visual grasping policies from static datasets. [16] gives a detailed comparison of fully offline training and online fine-tuning for such grasping tasks. The recent Conservative Q Learning (CQL)[18] approach is arguably the most successful offline RL approach to date, and directly addresses the extrapolation error phenomena by introducing a regularizer loss term that helps learn a conservative Q-function. They theoretically show that the expected value of the policy under this Q-function lower bounds its true value. In experimental evaluations, CQL achieves 2-5x higher return than the next best offline RL methods on the D4RL benchmark. Older examples of offline RL techniques

4.2 World on Rails

World on Rails (WOR) addresses this by using three key strategies

Algorithm 1: Learning in a world-on-rails

Data: Training trajectories D
Result: Policy $\pi(I) \in \mathcal{A}$
`// Forward-model fitting §4.2.1`
Function FitForward$(D) \rightarrow \mathcal{T}^{ego}$**:**
  Minimize Equation (4.1);
  return ego-vehicle forward model $\mathcal{T}^{ego}$;
end
`// Action-value estimate §4.2.2`
Function EstimateQ$(D, \mathcal{T}^{ego}) \rightarrow Q$**:**
  for $\tau \in D$ **do**
    Initialize $V_{|\tau|+1}(\cdot) = 0$;
    for $t = |\tau| \dots 1$ **do**
      Compute Q_t an V_t Equation (4.2);
      Store Q_t;
    end
  end
  return stored Q-values;
end
`// Policy distillation §4.2.3`
Function DistillPolicy$(D, Q) \rightarrow \pi$**:**
  Minimize Equation (4.3);
  return visuomotor policy π;
end
Learn forward model $\mathcal{T}^{ego} =$ **FitForward**(D);
Estimate action-values $Q =$ **EstimateQ**$(D, \mathcal{T}^{ego})$;
Learn visuomotor policy $\pi =$ **DistillPolicy**(D, Q);

1. Training a *forward model* to predict future ego agent states given actions
2. Leveraging *privileged information* only available in simulation to help compute action value functions in a tabular fashion.
3. Using *policy distillation* to train a visuomotor policy that maximizes expected return according to the tabular action value functions.

4.2.1 Forward model and the "rails" assumption

Model-based RL methods all train some form of forward model that, given a state and action, predicts future states. Intuitively, this approach roughly emulates human decision-making - one will often consider different actions and imagine their possible outcomes when planning a course of action. However, this can be an extremely difficult task depending on a number of factors. For example, if the state representation is complex (e.g. a bird's-eye-view image), it can be difficult to predict future states with accuracy. Another issue is the *multimodality* and uncertainty in other agents' decision-making processes. There are multiple ways other agents can respond to the ego agent's actions. For example, if the ego agent is waiting to make an unprotected left turn, oncoming traffic may choose to speed up and pass, slow down and let the ego agent turn, or even come to a complete halt. To what extent should we predict the behavior of other agents? One end of the spectrum of approaches is to create a dedicated prediction model that forecasts the trajectories of other agents over a number of different ego and/or other agent actions, but this is time-consuming and complex. This is a common approach in industry, where entire teams are formed to solve the prediction problem.

WOR addresses this in a simple and surprising manner. First, they decompose the forward modeling problem into the ego component and the non-ego components (e.g. other vehicles, pedestrians). The ego vehicle's state and dynamics are modeled according to the bicycle model, with the state at time t defined as $\hat{L}_t^{\text{ego}} = (x_t, y_t, v_t, \theta_t)$. A simple multi-layer perceptron network is sufficient to predict how the ego agent's state changes given an action, and self-supervised data are easily generated by running any vehicle agent in simulation and recording its position, orientation, and speed. The network is trained to minimize Equation (4.1). WOR's key innovation is the adoption of the "rails" assumption for the non-ego component. In essence, this assumption asserts that the rest of the world do not react differently even if the ego agent chooses a different action than what was recorded in the dataset. Non-ego vehicles and pedestrians are simply replayed as they were from the log.

$$E_{\hat{L}^{ego}_{t:t+T}, \hat{a}_t}\left[\sum_{\Delta=1}^{T} \left| \mathcal{T}^{ego\Delta}(\hat{L}^{ego}_t, \hat{a}_{t+\Delta-1}) - \hat{L}^{ego}_{t+\Delta} \right| \right], \tag{4.1}$$

Intuitively, this factorization of the forward model should not work. In the real world, it is clear that drivers and pedestrians will react to each other's actions. Otherwise, drivers and pedestrians would exhibit clearly unsafe behavior and run into each other with reckless abandon. How, then, can a safe driving policy be learned from such a seemingly dangerous assumption? The WOR authors assert that despite the unrealistic nature of the rails assumption, it can still provide strong and useful supervision to the agent in combination with the rest of the WOR approach. While the world may not react to the agent's actions, the agent should still be able to learn good driving behavior that translates to the real world e.g. giving other vehicles and pedestrians a wide berth, waiting for clearance before making an unprotected left, etc.

4.2.2 Computing a tabular action value function

Arguably the most important component of the WOR approach is the tabularized action value function and how it is computed. For long horizon planning problems, the standard approach is to estimate this using a deep neural network through some form of temporal-difference (TD) learning. WOR adopts a different approach that is made possible in part by the privileged information available in simulation. Concretely, they restrict the planning problem to have a relatively short terminating horizon of T timesteps. Events that occur past the planning horizon are not accounted for by any sort of cost-to-go estimation. Then, given a forward model and a reward function, the return for a given action (i.e., the Q value) can be explicitly computed without needing any approximations.

Consider the problem of computing a value function V_t given recorded ego state $\hat{L}_t^{\text{ego}}$ and world states $L_t^{\text{world}}, ..., L_{t+T}^{\text{world}}$. V_t is represented as a 4D tensor discretized into $N_H \times N_W$ position, N_v velocity, and N_θ buckets with $\hat{L}_t^{\text{ego}}$ centered in this discretization. Each surrounding 4D coordinate then represents a possible state configuration L_t^{ego} for the ego agent that can be used to index into V_t to retrieve the return of that configuration. Given a T-step horizon, we compute V_t by starting at time $t + T$ (with V_{t+T} being a zero tensor), and working backwards via dynamic programming. For all 4D state configuration coordinates represented in V_{t+T-1}, the forward model estimates how each configuration L_{t+T-1}^{ego} evolves given each of 28

possible actions as $\mathcal{T}^{ego}(L^{ego}_{t+T-1}, a_{t+T-1})$. The majority of these state/action pairs will generate state configurations L^{ego}_{t+T} that exist in V_{t+T}. If we have access to a reward function, we can compute the Q value for a particular state/action pair as $Q_{t+T-1}(L^{\text{ego}}_{t+T-1}, a_{t+T-1}) = r + \gamma V_{t+T}(L^{\text{ego}}_{t+T})$. Applying a max operator over all 28 Q values for each state L^{ego}_{t+T-1} computes the correct values for V_{t+T-1}. This process is repeated for V_{t+T-2} and onwards, until we reach V_{t+1} at which point we may compute the action values $Q_t(\hat{L}^{\text{ego}}_t, a_t)$ that are used to train the image policy for this timestep.

WOR sets up a reward function that considers ego and world state, ego action, and high-level Leaderboard command and gives a scalar reward that expresses how good the agent's state and chosen action are for the current world situation. For example, given a highway situation where the agent must drive straight on a specific lane, it should be rewarded for being in and aligned with the correct lane while moving with forward velocity. If there is a blocked agent in front, the ego agent should be rewarded for braking when approaching it. If the agent is asked to change lanes, it should be rewarded for actions that get it closer to the desired lane. Concretely, the reward function is expressed as $r(L^{\text{ego}}_t, L^{\text{world}}_t, a_t, c_t)$. Because of the rails assumption, we assume that all world states L^{world}_t play out exactly as recorded even if the ego agent takes a different action. While training, we simultaneously compute action values for every available high-level command c_t yielding, WOR's version of the Bellman backup equation as described in Equation (4.2)

$$\begin{aligned} V_t(L^{ego}_t) &= \max_a Q_t(L^{ego}_t, a) \\ Q_t(L^{ego}_t, a_t) &= r(L^{ego}_t, \hat{L}^{world}_t, a_t) + \\ &\quad \gamma V_{t+1}(\mathcal{T}^{ego}(L^{ego}_t, a)). \end{aligned} \tag{4.2}$$

The agent is given a +1 reward for staying in the target lane at the desired position, orientation, and speed, and is linearly penalized for deviating from the lane to a value of 0. Since we have access to *privileged information*, we can also recover ground truth traffic light state and the true positions and orientations of other traffic participants. This is used to set up "zero-speed" regions at red lights and near other vehicles and pedestrians. The agent earns a +5 reward for braking in a zero-speed zone, but this reward cannot be accumulated to prevent abuse. The authors find that zero-speed regions are sufficient for evoking collision avoidance behavior - collisions are

not explicitly penalized. We emphasize that this reward function is nigh-impossible to accurately compute outside of simulation, since we cannot access the ground truth information required without resorting to estimation via manual annotation or upstream modules, e.g. depth prediction, object detection.

4.2.3 Distilling a visuomotor policy

With action values computed for each training sample in the dataset, we can distill a visuomotor policy π that consumes a training image $\hat{I}_t$ and must predict a distribution over actions that maximizes the expected return according to the computed Q values. We also include an entropy regularizer H to encourage a more diverse output policy, with α being a temperature parameter. The optimization objective is formalized as

$$E_{\hat{L}_t^{ego},\hat{I}_t}\left[\sum_a \pi(a|\hat{I}_t)Q_t(\hat{L}_t^{ego},a) + \alpha H\Big(\pi(\cdot|\hat{I}_t)\Big)\right]. \tag{4.3}$$

4.3 Failure Analysis

While WOR exhibits the best performance out of all current Leaderboard methods, it still falls far short of achieving a perfect driving score. In this section, we will present a detailed analysis and discussion of the failure modes most common to the WOR image agent, assisted by our contributions to the evaluation toolkit. There is a crucial difference in WOR's evaluation compared to LBC's evaluation. Since WOR is a more recent codebase, it includes all ten scenarios described in the NHTSA pre-crash typology, while LBC only models three of the scenarios. Also, the WOR agent was benchmarked for 5 repetitions on each validation route, rather than 10 repetitions, as was done for the LBC image agent. This mostly manifests in the infraction pie chart (Figure 4.1), which counts the total number of infractions over all routes and repetitions.

As opposed to LBC, the WOR approach only trains a single agent - the final image policy that is submitted to the public evaluation server. Rather than using a privileged model, the tabular action value function is used to densely supervise the image policy during policy distillation. We find that the ego model is not a large

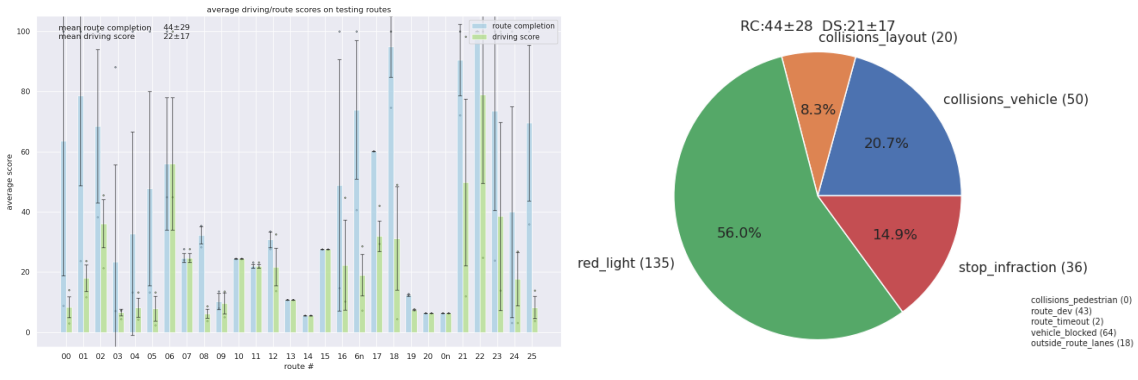


Figure 4.1: Baseline WOR average metrics

source of error because it is only ever used to predict bicycle dynamics one time step into the future, which is a relatively simple task easily solved by the multilayer perceptron model. During our failure analysis, however, we find that the computed action value functions exhibit errors that provide incorrect supervision to the image policy, causing it to frequently incur specific types of infractions. We proceed through the failure modes roughly in order of how much they affect the final driving score.

4.3.1 Route deviations

Route deviations emerge as the most pressing issue that World on Rails faces. While they are not visualized on the pie chart, each route deviation infraction immediately terminates the evaluation run. With 28 routes in the validation split, and 5 repetitions run per route, there are 140 total runs that the image policy performs. Of the 140 total runs, 43 of them - almost one third - ended in a route deviation termination. It is difficult to directly translate the route deviations into how they affect overall driving score, since there is no way of knowing how the agent would have performed/what infractions it might have incurred had it not deviated from the route. However, we can analyze individual routes to get a sense of why route deviations occur, and what effect it has on route completion scores.

On certain routes, the WOR agent will inevitably make the same exact mistake each repetition and end the run with the same route completion score. Route 13 (bottom image in Figure 4.2) is a simple example where the ego agent must progress along a highway route while occasionally changing lanes. However, the agent will

invariably take the same exit each time despite being told to continue straight along the highway. This causes a route termination and caps route completion at 10%, forgoing 90% worth of possible metric gain. Visually, the highway exit very gradually separates from the lane that the agent is meant to continue along. As opposed to LBC's waypoint-based control policy, the WOR agent computes an expectation over the image model's predicted action distribution which may contribute to its less decisive lane-keeping behavior in these situations.

Figure 4.2: Top: route 13 deviation example. Bottom: route 10 deviation example. In both images: left is wide-angle front-facing image, right is narrow-angle telephoto image

Other route deviations are caused by more complex situations that have many possible pitfalls and require sophisticated decision-making. We observe that the car-following cue again plays a significant role in misleading the ego-agent. Route 10 (top image in Figure 4.2)) spawns the agent in the middle of the highway, where it must progress towards an exit with a traffic intersection at the end. Leaderboard instructs the agent to make a left turn, but it must make the turn while yielding to cross-traffic coming from the left. Also, the lane that it must turn onto is backed up by traffic coming from the right so the agent must wait for it to clear before turning

into the lane. In every single repetition, the ego agent successfully avoids colliding with the moving cross-traffic, but moves towards the backed-up traffic and comes very close to colliding before completely switching to a right turn and following the cross-traffic, leading to a route deviation. While this particular situation is consistent across all repetitions of route 10, other routes contain multiple scenarios that are often difficult for the agent to navigate. On route 14, there are three particularly difficult intersections where the agent must make an unprotected left against oncoming traffic. In many cases, the agent attempts to execute the left turn but will swerve slightly to avoid oncoming traffic. If the agent moves too drastically while avoiding collisions, it will frequently align itself with the wrong lane and continue driving, leading to a route termination.

It is not immediately obvious what the best way to address this specific failure mode is due to how complex and inconsistent its causes are. One possible solution is to tune the reward function to incentivize staying in the correct route, but this may lead to greedy behavior where the agent runs lights or collides with vehicles in order to gain the route progression rewards. It may be more appropriate to apply heavier penalties to route lane deviations, rather than just decaying to 0.

4.3.2 Running red lights

As with LBC's image policy, the plurality of the WOR image policy's failures are traffic light infractions. This is again a surprising result, due to many of the same reasons that were discussed in the LBC failure analysis. Traffic light intersections have simple decision rules - go on green, slow down on yellow, and stop on red. Furthermore, WOR introduces a telephoto camera for the express purpose of capturing distant traffic lights more clearly and prominently. Why are red light infractions still so prominent among WOR's failures? We find that there are two primary reasons: (1) as before, yellow lights transition to red extremely quickly and (2) the action value functions are sometimes computed incorrectly at intersections.

Short yellow lights

As before, fast yellow light transitions are a major reason why the WOR agent causes infractions. Unlike LBC, however, it should be possible to directly incentivize agents

to stop or at least slow down at yellow lights with RL rewards. Nevertheless, we observe that the baseline agent often does not slow down at yellow lights and will continue into the intersection. This seems to be especially true when there is a car in front of the ego agent that has already entered the intersection, again hinting at the influence of the car-following cue. One hypothesis on why this happens is that the planning horizon for action value computation may be too short. If it takes 3 seconds for a yellow light to turn red, then any training samples drawn from the first 1.75 seconds of the yellow light will fail to recognize that the light is about to turn red. While computing tabular action values via dynamic programming, these samples will not see any reward for braking in the red traffic light's no-speed zone and will likely assign high values for actions that continue into the intersection and collect the route progression reward. This suggests that extending the planning horizon is one possible solution for this issue.

Incorrect action values

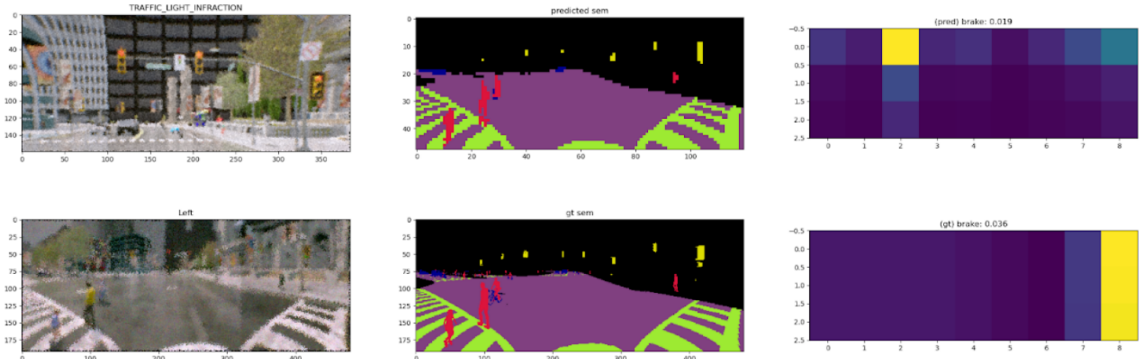


Figure 4.3: Ground truth action values do not recommend braking despite red light

We find that there are errors in the way that the ground truth is computed for traffic light intersections. While training, many data samples at traffic intersections will be visualized similarly to Figure 4.3. Here, we see that despite the traffic light clearly signaling red in the telephoto image, the ground truth action value distribution assigns nearly zero value to braking. This is clearly an incorrect cue that leads the agent into a red light infraction, and often causes collisions and/or route deviations. Since half of each training batch is composed of infraction samples, having incorrect

action values for these infraction situations would certainly degrade performance if the issue is prevalent. A significant number of red light infractions in the dataset exhibit this phenomenon. That is not to say that all red lights have incorrect action values, but traffic intersections that did cause infractions also happen to have incorrectly computed values.

It is not immediately clear why the ground truth computes such an incorrect value. Agents are incentivized to brake when traffic lights are red, although this reward can only be given once to prevent abuse. Agents are also not penalized for moving in a red light no-speed zone, unlike non-ego vehicle no-speed zones. One possible explanation might be how the traffic light penalty regions are positioned - if they do not extend far into the intersection, it may actually be more worthwhile for an agent that is almost in the intersection to simply continue moving (penalty-free!) and collect the route progression reward. Nevertheless, Table 6 in the World on Rails paper shows that the approach does appear to suffer from generalization issues and experiences almost no traffic light infractions on NoCrash training towns while experiencing significantly more infractions on test towns. This implies that besides having occasionally incorrect traffic light action values, the WOR image model may also be overfitting to the training set or learning to use the wrong cue at intersections.

4.3.3 Vehicle collisions

While less prevalent in WOR than in LBC, vehicle collisions still significantly contribute to the WOR agent's penalty multiplier. We observe that the agent is very good at avoiding collisions with static vehicles or other vehicles moving in the same direction. One common failure mode we will quickly note occurs when the baseline agent attempts to change lanes. The triple camera setup affords the agent a wide field of view, but there is no perception for the rear view. In other words, the agent makes all lane changes without any idea of what is behind it. This often leads to side swipes where it will brush against the side of a passing vehicle in the next lane. The most obvious solution for this is to integrate an additional sensor to give rearview perception - perhaps a LiDAR or a backwards-facing camera. Besides lane changes, the agent also exhibits poorer collision avoidance at night. For example, the baseline does not make any collisions on route 16 during the day, but hits multiple vehicles

when running the same route at night.

The dominant vehicle collision failure mode occurs when the agent must navigate intersections while avoiding high-speed vehicles and cross-traffic. We previously noted that red light traffic infractions are a common upstream cause of vehicle collisions due to dangerous cross-traffic that enters the intersection. Leaderboard also includes many situations at intersections where the agent must make an unprotected left turn or a right turn on a red light. Consider route 18 (Figure 4.4): towards the beginning of the route, the agent must wait for cross-traffic to clear before making a right turn. Despite seeing a sedan speeding across the intersection from the left, the agent will often make the turn and collide with the sedan, as shown below. In another case, the agent has the right of way and proceeds straight across an intersection, but gets hit by a CyberTruck that blows past the red light. Even though it can clearly see the Cybertruck entering the intersection, the agent makes no attempt to avoid collision.

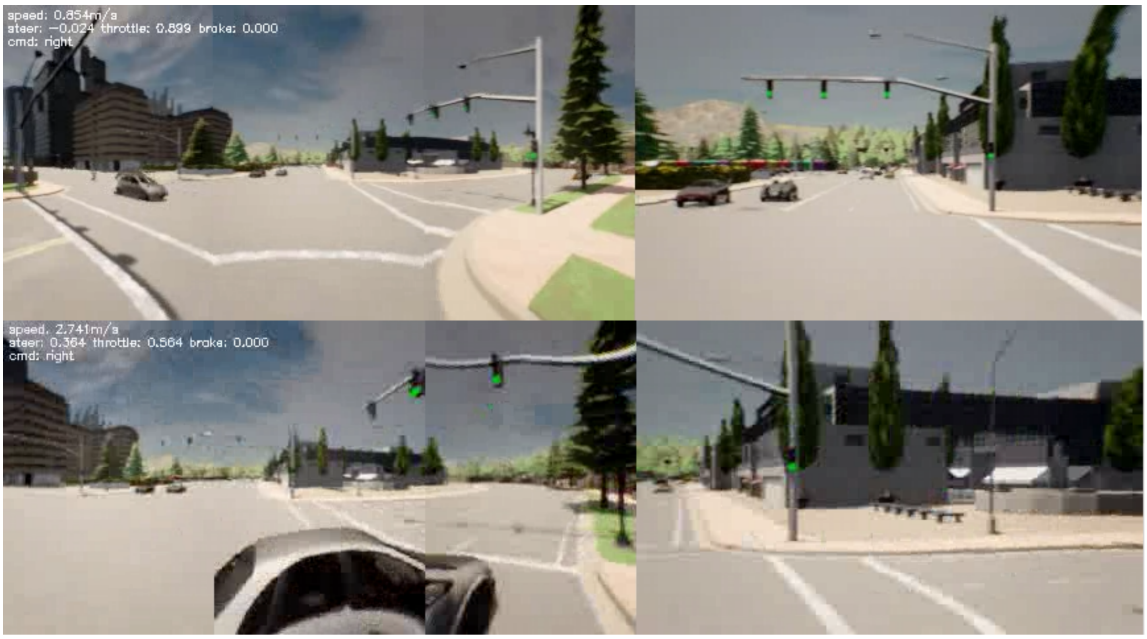


Figure 4.4: Top: sedan illegally enters intersection from the left. Bottom: baseline agent runs straight into the sedan.

We think this is caused by the short planning horizon issue that was described in Section 4.3.2. In both collisions, the other vehicle was moving at very high speed prior to impact. Consider how the action values for such a situation would be computed. If

the other vehicle starts relatively far from the ego agent but is quickly moving in its direction, a short planning horizon might fail to see that moving into the intersection is dangerous. Given only 1.25s to plan ahead, the other vehicle might appear to be far enough to disregard when in reality it is on a direct collision course with the ego agent. Extending the planning horizon could allow the vehicle's no-speed reward zones to influence the action values and improve collision avoidance with a simple hyperparameter change.

4.4 Technical Approach

Online reinforcement learning approaches iterate very finely between data collection and policy optimization, but it is computationally intensive and time-consuming to run (often multiple) simulations while also training a neural network at the same time. On the other hand, offline reinforcement learning approaches completely eschew interaction with the environment after data collection, but this is a fairly restrictive approach in practice. For example, self-driving car companies will deploy their latest software and models on autonomous fleets to gather new data, and use the new data to make further changes to their software stack before redeploying their updated fleets for further evaluation and data collection in an iterative development process.

4.4.1 SORL

With this observation, we propose a middle-ground approach between offline and online reinforcement called "Slightly Online Reinforcement Learning" (SORL). SORL adopts a similar workflow by reintroducing on-policy data collection to offline RL methods as an additional stage, forming a two-stage approach that can be iterated. Concretely, after training an agent via offline reinforcement learning (e.g. WOR in this project), we collect a new dataset using the agent and retrain, optionally mixing in data from previous iterations. While WOR does an excellent job of learning strong policies from a static dataset, the distilled image policy is still only trained on images and situations encountered by the privileged data collection agent. At test time, the WOR agent will inevitably stray into situations that it did not see during training. While the model may be robust enough to handle some of these situations, it is still

disadvantaged by the inability to use these situations for further training.

We do not claim any special novelty to this training schema - it is heavily inspired by DAgger and addresses the same distribution shift issues present in offline imitation learning. This approach can also be seen as a coarser interpretation of the standard online RL loop which finely interleaves policy optimization and data collection. By separating data collection and policy optimization into individual stages, SORL strikes a middle-ground approach between entirely offline and online approaches, hence the "slightly online" moniker. This approach is simple to implement with any offline RL algorithm and resembles the workflow already used in the self-driving industry.

Due to time and compute constraints, we are only able to run a single iteration of SORL. We do not have access to the WOR dataset due to its size (1M frames taking up 4.3 Terabytes), but we do have access to the trained WOR weights courtesy of the authors. The pretrained WOR weights are used both as the data collection policy and the initialization for the next model to be trained. One practical issue is that it takes longer for data collection agents using the WOR model to deploy, since every planning step requires inference through a deep neural network, as opposed to the hard-coded autopilot agent. We run the SORL data collection agent on all 52 training routes for 4 repetitions each, with each run terminating on collision or route completion. Our final dataset contains about 116K frames gathered at 4 Hz, which is equivalent to around 8 hours of driving. Data augmentations effectively triple this to around 24 hours of driving. We also experiment with three experimental modifications to the training procedure described in the next subsections.

4.4.2 Learning without forgetting

Since we don't have access to the original WOR dataset, there is a risk that the model will overfit to the SORL dataset and forget what it learned previously. One standard solution for learning without forgetting is to use the predictions of the initial baseline model π^* as pseudo-labels. The student model is then penalized for deviating from the baseline model's predictions. This is conceptually very similar to the expert imitation loss used in Chapter 3. Concretely, we minimize the KL divergence between the student model and the baseline model's action distribution. With hyperparameter

β weighting the KL divergence loss, we modify Equation (4.3) as

$$E_{\hat{L}_t^{ego},\hat{I}_t}\left[\sum_a \pi(a|\hat{I}_t)Q_t(\hat{L}_t^{ego},a) + \alpha H\Big(\pi(\cdot|\hat{I}_t)\Big) + \beta D_{\mathrm{KL}}\left(\pi(\cdot|\hat{I}_t)||\pi^*(\cdot|\hat{I}_t)\right)\right]. \quad (4.4)$$

4.4.3 Prioritized replay

We also carry over the prioritized replay experiments from the previous chapter to this approach. LBC/DQN was able to discourage incorrect expert actions, but could not determine which actions were better to take. On the other hand, WOR is able to provide supervision on the *correct* alternate actions to take in infraction scenarios. We expect prioritized replay to have a greater effect on fine-tuning WOR's infraction issues. Similar to before, we record any infractions that occur during data collection. A sample is considered an infraction sample if an infraction occurs in the next T steps, where T is the planning horizon. We define a parameter κ that is the proportion of each training batch containing infraction samples (sampled uniformly). For SORL, κ is not annealed and is held constant throughout training.

4.4.4 Extended planning horizon

Finally, we experiment with a simple change to the planning horizon parameter T by simply increasing it from 5 to 12 timesteps, or from 1.25 to 3 seconds. In practice, this causes the action value computation stage to take longer by a factor approximately equivalent to the relative increase in planning horizon. However, this stage only needs to be run once per policy optimization, stage making this mostly negligible. Extending the planning horizon helps the action values see further into the future, hopefully addressing the issues with yellow lights and high-speed collisions discussed in the failure analysis.

4.5 Experiments

We train each model on the dataset for 10 epochs and choose the final model for benchmarking. We use $T = 5$ for the non-extended planning horizon and $T = 12$ otherwise. We use $\kappa = 0.01$ for prioritized experience replay. Originally, our infractions

dataset contained 366 samples, but this is reduced to 55 because we remove the red light infractions due to incorrect action value computation. Note that the "PER with expert imitation" results were computed before we uncovered the incorrect action values phenomenon, so that the model is trained with the red light infractions.

For evaluation, we run each proposed agent on the Leaderboard validation routes for 5 repetitions each and record the same metrics as before. There are two extra routes, which are simply routes 16 and 20 run at night time. Importantly, WOR evaluation uses an updated set of scenarios that are triggered as the agent moves along the validation routes. LBC's evaluation only included scenarios 1, 3, and 4, which are relatively simple situations. Besides the "ControlLoss" scenario, which doesn't appear to really do anything, the other two scenarios are variations on a pedestrian or biker illegally crossing the street. On the other hand, WOR evaluation includes all 10 NHTSA scenarios, which are much more challenging scenarios. These include unprotected left turns, blocked leading vehicles, and hazard signs thrown onto the road. This means that WOR benchmark results cannot be directly compared to the LBC results, since the WOR evaluation is more difficult. However, there are more pressing issues specific to the WOR benchmark that affect how we can interpret the average route completion and driving score metrics previously used in the LBC evaluation.

4.5.1 Benchmark Issues

The most serious issue is that some route deployments are terminated *not* due the ego agent's actions, but because of Leaderboard errors or completely unknown issues. One example is found on route 6 (right image on Figure 4.5) which consists of mostly highway driving. Partway through the route, agents will stop and not move forward for no apparent reason at all, causing blocked-agent timeouts unless it disobeys Leaderboard's "follow-lane" command and changes lanes. Another case is found in the beginning of route 0 (left image on Figure 4.5), where there will always be backed-up traffic that is stopped. We can look at the WOR agent's telephoto camera and see that there is a road hazard sign that is partially blocking the road. This is actually a Leaderboard scenario that tests the ego agent's static collision avoidance. However, this has the unintended effect of causing other agents on the road to come

to a complete stop.

Figure 4.5: Left: blocked traffic on route 0. Right: blocked traffic on route 6.

There are generally two outcomes in this situation: (1) the blocked non-ego agent despawns and the remaining traffic and the ego agent are able to proceed with the route or (2) traffic is blocked for so long that the ego agent is terminated due to route timeout even though *it did nothing wrong*. These two outcomes seem to happen randomly, which has the unfortunate effect of making average route completion score and driving score unreliable metrics of driving performance. If we compared two agents' average metrics on route 0, we might say that agent A is better than agent B when in reality agent B just got unlucky and saw more blocked traffic situations.

Another more subtle issue is how the "SignalJunctionCrossingRoute" scenarios are triggered. These are very challenging situations where the agent must cross or turn in an intersection while cross-traffic is moving laterally. This can be something like an unprotected left turn, a right turn on red, or even a simple straight intersection crossing with cross-traffic illegally running red lights. However, these scenarios are only triggered when agents are a specific distance from the intersection. Consider the route 14 intersection shown above in Figure 4.6. On the left, agent A stops despite the green light and times out due to inactivity. On the right, agent B also stops at the green light but *sometimes* inches forward before stopping again. This triggers cross-traffic that illegally runs the intersection, but this is sufficient to activate the car-following cue and allow agent B to cross the intersection and finish the route. Agent A would likely have done the same thing if the cross-traffic scenario activated, but this never happens since it doesn't randomly inch forward like agent B. Agent A would then perform worse according to the average driving score/route completion score metrics, but we cannot claim that agent B had better performance on this

Figure 4.6: Left: blocked agent does not trigger scenario. Right: blocked agent that inched slightly forward triggers cross-traffic

specific route since it also erroneously stopped at a green light. Agent B simply got lucky and took advantage of the Leaderboard scenario triggered by its erratic and inconsistent forward movement. This hints at a more fundamental problem with using driving score as a metric representation of good driving behavior that is further discussed in Section 5.1. While these sorts of issues do not appear on every route, we must be cautious in our analysis and treat average metrics as very general measures of driving performance with this context in mind. We will instead focus on supporting claims of improved behavior with *qualitative* examples.

4.5.2 Results

Vanilla SORL

The most basic approach is to simply train on data gathered from the student's state distribution with no additional training modifications i.e., "vanilla" SORL. Compared to the baseline, this approach yields an agent that achieves a lower average route completion score, but maintains almost the same driving score. The infraction distribution also appears to show a reduction in red light infractions, which seems to be desired behavior. However, we cannot attribute this to an actual change in behavior because the reduction in red light infractions is often due to this agent's tendency for early terminations on routes that contain many red lights. Overall, the reduction in average route completion is the primary reason for the corresponding reduction in the number of penalty-inflicting infractions. This can be explained in

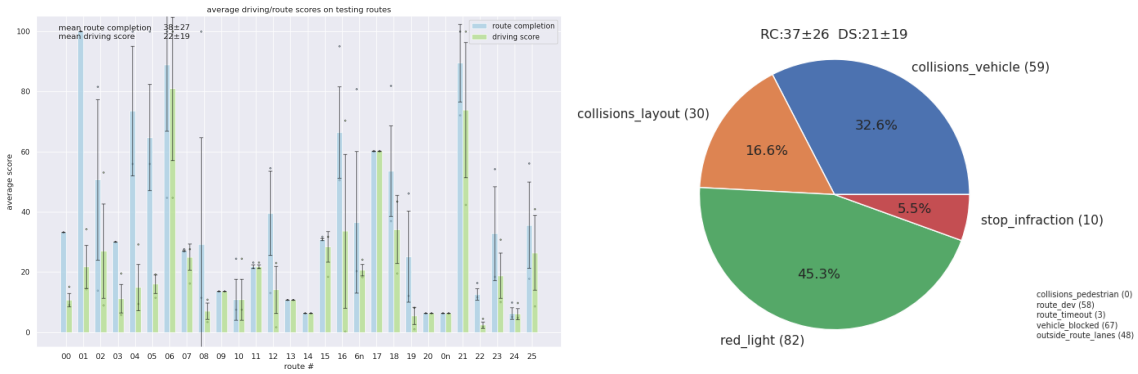


Figure 4.7: Vanilla SORL results

part by the significant increase in the incidence of "outside route lanes" infractions and a corresponding increase in route deviations, suggesting that the proposed model is more likely to be led astray. This is especially evident when you compare per-route average driving score and route completion scores between this approach and the baseline

Why do we see such a decrease in performance compared to the baseline? Qualitative analysis of evaluation videos and per-route performance charts suggest an across-the-board reduction in "good" behavior - e.g. changing/maintaining the correct lane, crossing intersections at the right time, and giving other vehicles a wide berth to avoid collisions. We believe this can be attributed to overfitting - the agent is trained on data gathered using the baseline WOR model, which significantly differs in size from the autopilot dataset that the baseline was trained on. Our dataset is about an eighth of the size of the original WOR dataset. Without this dataset on hand, additional training will cause models to drift away and forget what the baseline model has already learned. At test time, this results in less robust and stable performance compared to the baseline in the absence of of additional modifications to the training schema.

PER with expert imitation

We see that combining prioritized replay with expert imitation and masking performs worse across the board compared to the baseline approach. At first look, this is a surprising result - we account for lack of access to the original dataset with expert

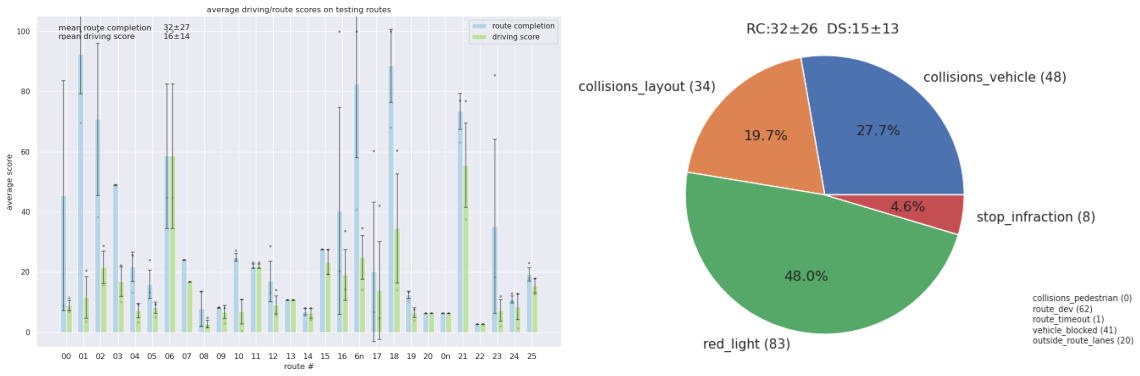


Figure 4.8: PER with expert imitation

imitation and focus on fine-tuning behavior in infraction situations using prioritized replay and expert masking. However, this significantly reduces the average route completion and driving score, while also generally reducing per-route performance for both metrics when compared to the baseline. The infraction pie chart (Figure 4.8) shows a comparable number of collision infractions and a higher incidence of red light infractions compared to simply training on additional data with no modifications, despite achieving on average 5% less route completion. This implies that prioritized replay might actually be worsening the agent's performance for these infraction situations, which is the opposite of what is meant to occur. Why would this be the case?

It turns out that these particular results suffered from the incorrectly computed action values we observed during failure analysis. Using prioritized experience replay, the majority of infraction samples demonstrated traffic light infractions which we previously showed to have unreliable ground truth action values. The model is then shown a disproportionately high amount of samples at training time, which inevitably translates to performance degradations at intersections, which are already the most common location for route deviations, vehicle collisions, and of course red light infractions. Qualitatively, we see this reflected in the proposed model's behavior at traffic lights. Consider the average infraction charts on route 2 compared to the baseline (Figure 4.9): despite achieving similar mean route completion scores, the proposed model averages more than twice as many red light infractions for the same average driven distance.

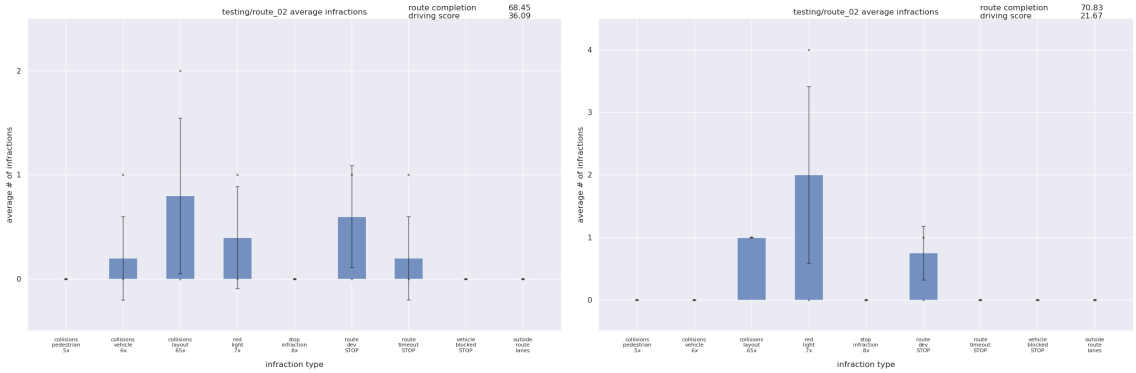


Figure 4.9: Left: baseline average infractions on route 2. Right: expert imitation and PER model average infractions on route 2.

Extended planning horizon

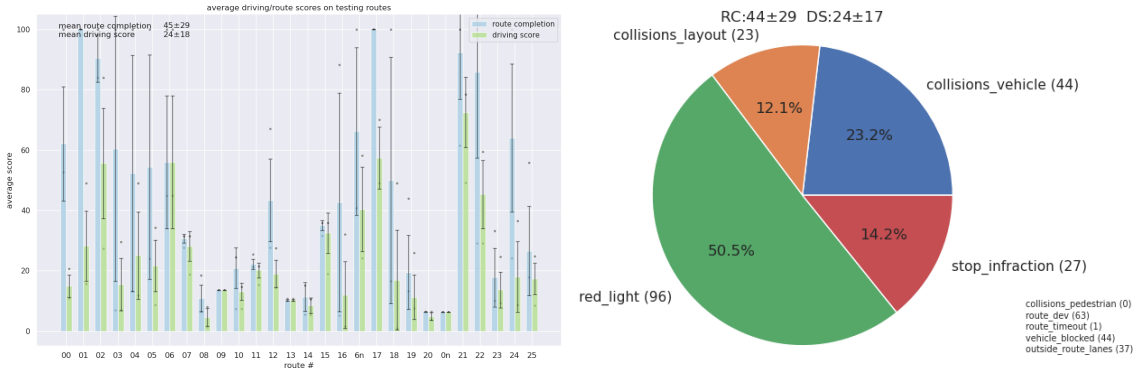


Figure 4.10: Extended planning horizon results

Simply extending the length of the planning horizon during computation of the action values appears to be the most straightforward and promising individual approach to improving driving performance, on the validation set. Despite not having access to the full dataset and training on the smaller local dataset, the agent maintained the same route completion score as the baseline WOR agent, but increased driving score by 3 points. As stated previously, this in of itself does not imply better driving performance but the infraction pie chart shows an approximately 30% reduction in red light infractions and a slight reduction in vehicle collisions, while increasing route deviations by 50%. Since the per-route route completion scores are similar to those of the WOR baseline, extending the planning horizon does appear to cause real

changes in driving behavior that help reduce these infractions. We can verify this by considering qualitative examples.

Passing a blocked car

One common situation is the "slow/blocked car" scenario that is depicted as Traffic Scenarios 5 and 6 in the NHTSA typology. In these situations, a car leading in front of the ego agent is either moving extremely slowly, or is completely blocked. To circumvent this, the ego-agent must change lanes to pass the car and continue along the route. Consider the difference in driving score and route completion score on route 17 between this approach and the baseline approach. The baseline routinely fails to progress past 60%, yet the proposed approach averages a perfect 100% route completion. Towards the end of route 17, there is a long stretch of highway that has a blocked car in the route lane.

At test time, the baseline agent approaches the car but never changes lanes to pass it, leading to a blocked vehicle termination of the run. However, extending the planning horizon causes the image agent to change to the right lane and avoid the car, allowing it to complete the route. Executing a lane change from behind a blocked car requires the ego agent to actually move closer to the blocked car for a short period of time before it can pass. Given a 1.25-second planning horizon, the computed action values may not see far enough into the future to see any benefit to changing lanes. However, extending the horizon to 3 seconds reveals that the agent will take a short-term penalty for moving in the blocked agent's "no speed zone" before benefitting from continued route progression after changing lanes. It should be noted that this behavior is not always executed perfectly - sometimes the agent will be stuck mid-lane change and will not progress unless other cars pass it and trigger the car-following cue. On route 20, this approach performs worse than the baseline in a similar blocked car situation because it will either get stuck or accidentally run into the car and trigger a collision infraction. We believe that this can be resolved with appropriate horizon tuning.

Obeying traffic lights

By extending the length of the planning horizon during action value computation, we observe a reduction in traffic light infractions on most routes. To best compare this

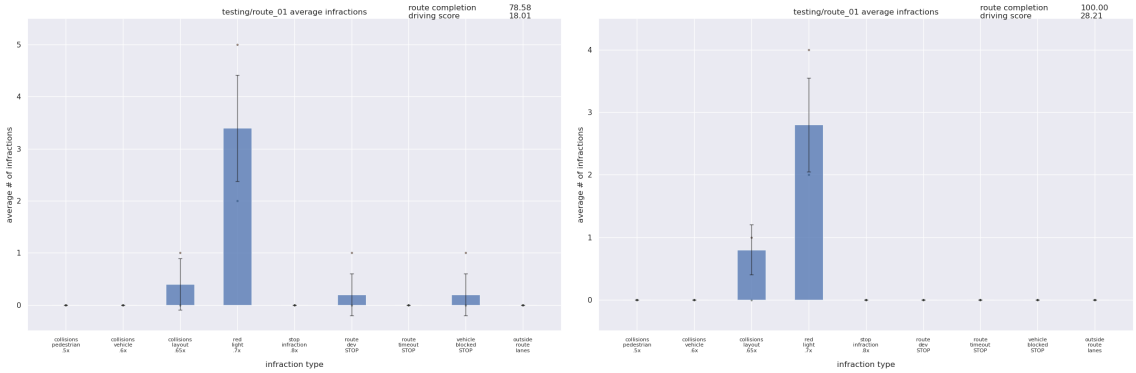


Figure 4.11: Proposed agent on right averages less red light infractions despite fully completing the route.

approach to the baseline approach, we look for validation routes where both agents performed similarly on route completion score and identify the incidence of red light infractions. Consider the average infraction plots for route 1 shown in Figure 4.11. Despite progressing 28% further along the route and encountering more traffic lights on average, the agent trained with extended planning horizons averaged less than 3 traffic light infractions per repetition while the baseline model averaged more than 3. However, this route suffers from the non-ego blocked agent issue, so we observe this phenomenon in a more granular fashion by comparing route performance charts on route 4, as shown in Figure 4.12.

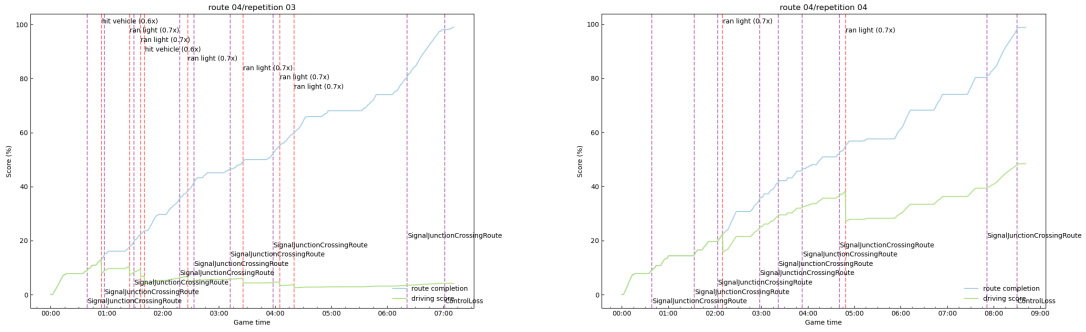


Figure 4.12: Left: baseline runs many traffic lights on route 4. Right: proposed agent only runs two lights.

In one run, the baseline approach incurs 6 traffic light infractions, which also

causes a vehicle collision in one instance. On the other hand, the extended planning horizon agent only suffers two traffic light infractions and completes the route with about an order of magnitude higher driving score. This pattern is consistent across all repetitions where the baseline and proposed approach achieved nearly 100% route completion. Qualitatively, we observe that the extended horizon agent differs in behavior from the baseline model on how to handle yellow lights. The baseline agent does not appear to be wary of yellow traffic lights and will enter the intersection, often leading to an infraction due to how quickly yellow lights turn to red. The extended horizon agent, on the other hand, generally starts braking when it observes a yellow light and will wait until the next green light before proceeding into the intersection. One hypothesis: when computing action values for yellow light traffic intersections, the 1.25s planning horizon may not be long enough to capture the subsequent red light and the corresponding brake reward incentives.

Extended planning horizon with expert imitation

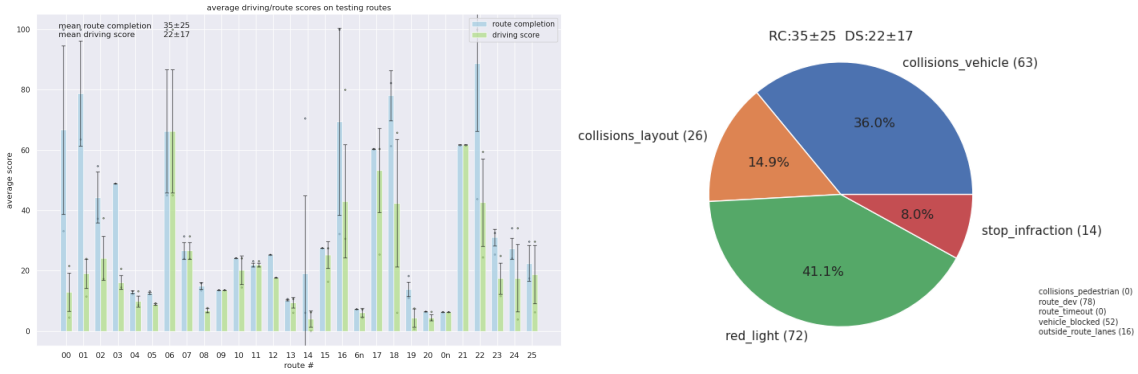


Figure 4.13: Extended planning horizon with expert imitation results

Extending the planning horizon and including expert imitation produces an agent that achieves a lower average route completion score than the baseline. It appears that expert imitation does not actually help with "learning without forgetting" because the number of route deviations is similar to the experiments that just train on additional data, and just extend the planning horizon. Additionally, the expert baseline model seems to pass on its own bad behaviors to the student model. Compare the performance on route 0 of this model to the model trained with just an extended

planning horizon (Figure 4.14). The proposed model generates more than twice as many red light infractions despite using the same planning horizon. Notably, the proposed model's performance profile resembles the baseline model's, which also had a significant number of red light infractions. Qualitatively, the proposed model shows similar behavior to the baseline model and has less of a tendency to slow down or stop for yellow lights.

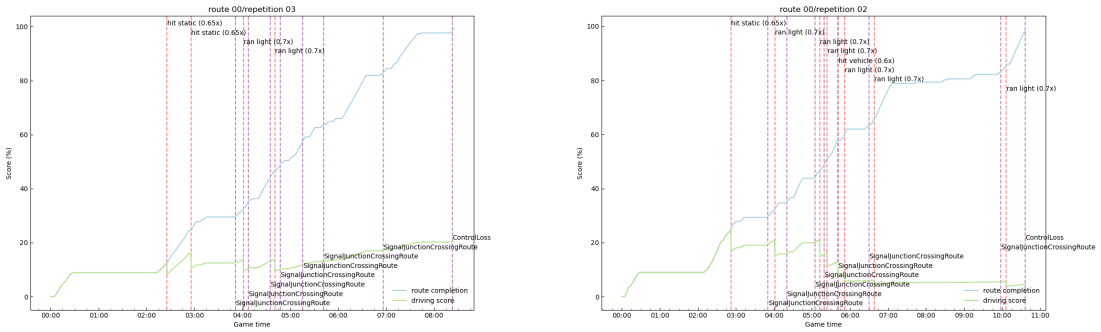


Figure 4.14: Left: extended horizon agent runs red lights less frequently than expert. Right: introducing expert imitation causes agent to revert to expert's red light running behavior

Extended planning horizon with prioritized experience replay

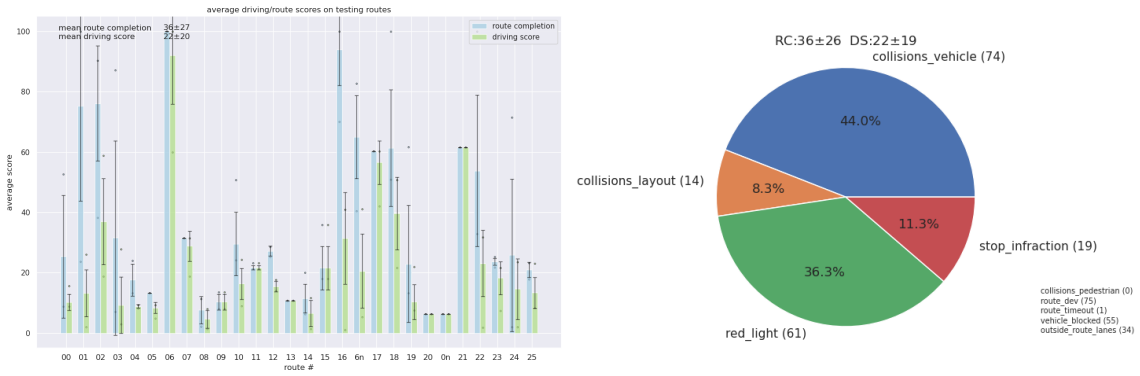


Figure 4.15: Extended horizon with PER results

Extending the planning horizon while using prioritized experience replay reduces average route completion score and driving score. It also surprisingly increased the

number of collision infractions, which runs counter to the whole point of the PER mechanism. We believe this is a clear example of overfitting on the infraction samples. Removing traffic light infractions from the dataset leaves just 55 collision infractions (165 with data augmentations). These samples are almost certainly overrepresented considering their relatively large proportion in each training batch. suggesting that more training data are needed to draw a conclusion about the efficacy of the PER mechanism.

4.6 Discussion

Our results show that WOR's performance is sensitive to the planning horizon hyperparameter. Simply increasing it from 1.25 to 3 seconds showed qualitative improvement in traffic light and collision avoidance and adds very little overhead to training time. Expert imitation does not seem to encourage "learning without forgetting", as each experiment that used it still had many route deviations. Prioritized experience replay also did not improve performance, but we do not think we can draw a conclusion about its efficacy given the extremely small number of infraction samples available.

Generally, this implementation of SORL does not seem to clearly improve upon WOR. However, logic suggests that SORL should improve upon completely offline approaches simply because it assumes access to the simulation and can train on more data. Our experiments were weakened by two issues: the lack of access to the original WOR dataset, and the inability to gather a similarly large dataset. We believe the SORL-generated dataset was too small and prone to overfitting, which may be one drawback of SORL - if policy models are large and slow on inference, the data collection stage may add significant overhead to overall training time. It may be interesting to see if SORL can train a model from scratch that matches/exceeds the baseline WOR model's performance using smaller amounts of data (compared to the 4.3 Tbs of WOR data), applied over multiple stages. The SORL method itself may also be improved by adding random exploration to each data collection stage to create even more diverse data. Future work should also experiment with the loss regularizer term introduced by CQL during policy optimization, as it is described as a simple modification to offline RL methods that yields significant improvement.

Chapter 5

Conclusion

5.1 CARLA/Leaderboard Critique

Compared to the previous NoCrash and CoRL2017 benchmarks, we think Leaderboard offers a significant improvement in evaluating good driving behavior. We are especially impressed by the NHTSA pre-crash scenarios that are modeled in the various towns and routes. We commend the CARLA team for their efforts in creating what is likely the most advanced and accessible open-source driving simulator available to the broader research community.

However, there are still many issues with Leaderboard and CARLA in general that must be addressed. The most obvious issue is that despite being dubbed a "photorealistic" driving simulation, CARLA is clearly very far from looking like the real world. While the weather, lighting, and sensor/physics simulation is very advanced, the assets within each town leave much to be desired. Vehicles, pedestrians, buildings, and natural objects do not have a significant amount of detail and (in the author's humble opinion) stray into the "uncanny valley" of photorealism. There is also a distinct lack of visual variety: there are fewer than 20 models of cars, pedestrians are of similar build/height and lack variety beyond simple permutations of hair/skin tone/accessories, none of the towns model dense city environments, etc. To be fair, this problem is not specific to CARLA - all simulation suffers from the `sim2real` issues. However, the state-of-the-art in computer graphics research or even recent triple A video games far outstrips what is seen in CARLA. A lack of photorealism makes

it difficult to put stock in benchmark results, since we cannot be sure they reflect real-world performance. A theoretical sensorimotor agent that completely solves every CARLA benchmark would almost certainly fail to show similar performance in the real world.

Specific to Leaderboard, we have already described some issues with the way certain scenarios are set up and triggered in Section 4.5.1. Another serious issue is a complete lack of pedestrians in any routes (outside of triggered scenarios), which clearly does not reflect reality. We also observe a lack of visual diversity in Leaderboard scenarios. For example, in LBC's Leaderboard evaluation there is a scenario where a pedestrian will illegally cross the road in front of the agent. In every single instance, the pedestrian spawns behind the same red vending machine (that also spawns/despawns out of nowhere) and runs across the road in the exact same way. This is a very simplistic interpretation of the corresponding NHTSA scenario that could theoretically be gamed. Rather than recognizing the danger of hitting the pedestrian, a model could learn to just stop when it sees the red vending machine and continue when it despawns. Similarly, WOR's implementation of the "blocked vehicle" scenario spawns the exact same green SUV each time. Luckily, most of these issues are relatively minor and could easily be fixed with additional work.

The most fundamental issue with Leaderboard is the following observation: *good driving score is not always an indicator of good driving performance.* We look to the performance of two agents on route 15 as an example (Figure 5.1). This is a highway driving scenario that exhibits the common "misleading exit" route deviation failure mode. When the exit lane splits from the highway lane, agent A invariably drifts to the right and follows the exit which terminates the run. Agent B never takes the exit and progresses along the route to completion. Leaderboard considers agent B to have better behavior because it achieved a higher driving score. However, it is misleading to say that agent B shows concretely better behavior for this route. Early on, the agent must make a right turn to get onto the highway. Agent A executes the turn from the rightmost lane of the intersection and ends up in the rightmost lane of the highway, which is the correct behavior. On the other hand, agent B makes the same right turn and ends up on the second lane from the right. Later on, it changes to the left lane in order to avoid some blocked traffic despite being told to follow the lane. Both of these actions should actually be considered incorrect behavior, since

agent B did not follow the Leaderboard route commands. Nevertheless, this sequence of events actually helps agent B down the line by making it very easy to avoid the highway exit.

Figure 5.1: Right: agent A erroneously takes the highway exit, leading to route deviation termination. Left: agent B avoids the exit by being in the wrong lane.

Since Leaderboard does not penalize agent B for not being in the correct lane, it achieves a higher driving score than agent A. We cannot know how agent B would have reacted in the more difficult situation that agent A was in, so can we concretely say that agent B has better driving behavior? Consider a thought experiment where this highway exit was removed from the route - it is entirely possible that agent A might have better handled the rest of the route than agent B. For this route, it does not feel right to judge agent A as worse purely based on driving score because it might actually be a stronger agent overall.

It is not immediately clear how to address this because the question of how to compute a metric that best represents "good driving" is itself a subject of debate. One possible solution for Leaderboard could be to not terminate agents when they are blocked or deviate from the route and instead respawn them at the closest part of the route, with the number of resets recorded as a secondary metric. A more comprehensive solution might be to completely drop the long, route-based evaluation that Leaderboard attempts and to focus exclusively on crafting scenarios that are diverse and difficult as a supplemental challenge benchmark to stress test agents. A candidate agent could be simply given a pass or fail score on each scenario with the goal of passing as many scenarios as possible. This would take advantage of the safety of testing in simulation while also avoiding many of the problems we see with driving score in Leaderboard.

5.2 Concluding Remarks

As expressed in Section 1.2, we set out to answer the following questions: (1) what is the current state-of-the-art in simulated driving, and (2) what value does simulation provide for the development of these algorithms? In the course of this work, we give a detailed review of Learning by Cheating and World on Rails which respectively represent the state-of-the-art in imitation learning and reinforcement learning for simulated driving. Despite their strong driving performance on CARLA benchmarks, we show that both methods still exhibit many basic failures that even inexperienced human drivers would never make. However, both methods perform far stronger than the prior state-of-the-art because of their use of *privileged information* to help provide strong supervision to image agents. Because of this, we propose that simulation should be a more integral part of the toolkit for training machine-learned robotics models. On the evaluation side, we show that the current CARLA benchmark still has many critical problems that make it an unreliable indication of good driving performance. We still emphasize that simulation should play an important role in evaluation and propose the creation of a challenge benchmark exclusively focused on difficult scenarios to supplement real-world testing. Despite its issues, we cannot ignore the benefits that simulation affords for the development and evaluation of robotics algorithms and we hope its capabilities and attention in the research community continues to grow.

Bibliography

[1] Carla documentation. URL https://carla.readthedocs.io/en/0.9.12/. 2.1

[2] Carla autonomous driving leaderboard, Feb 2020. URL https://leaderboard.carla.org/. 2.3.2

[3] Tanmay Agarwal. On-policy reinforcement learning for learning to drive in urban settings. Master's thesis, Carnegie Mellon University, Pittsburgh, PA, August 2020. 3.1.2

[4] Hitesh Arora. Off-policy reinforcement learning for autonomous driving. Master's thesis, Carnegie Mellon University, Pittsburgh, PA, August 2020. 3.1.2

[5] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016. 3.1.1

[6] Chenyi Chen, Ari Seff, Alain Kornhauser, and Jianxiong Xiao. Deepdriving: Learning affordance for direct perception in autonomous driving. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2722–2730, 2015. 3.1.1

[7] Dian Chen, Brady Zhou, Vladlen Koltun, and Philipp Krähenbühl. Learning by cheating. In *Conference on Robot Learning*, pages 66–75. PMLR, 2020. 3.1.1, 3.2

[8] Felipe Codevilla, Matthias Miiller, Antonio López, Vladlen Koltun, and Alexey Dosovitskiy. End-to-end driving via conditional imitation learning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–9. IEEE, 2018. 3.1.1

[9] Felipe Codevilla, Eder Santana, Antonio M López, and Adrien Gaidon. Exploring the limitations of behavior cloning for autonomous driving. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 9329–9338, 2019. 2.3.1, 3.1.1

[10] Alexey Dosovitskiy, Germán Ros, Felipe Codevilla, Antonio M. López, and Vladlen Koltun. CARLA: an open urban driving simulator. *CoRR*,

abs/1711.03938, 2017. URL http://arxiv.org/abs/1711.03938. 2.1, 2.3.1

[11] Uber Elevate. Fast-forwarding to a future of on-demand urban air transportation, 2016. URL https://evtol.news/__media/PDFs/UberElevateWhitePaperOct2016.pdf. 1.1

[12] E. Espié, Christophe Guionneau, Bernhard Wymann, Christos Dimitrakakis, Rémi Coulom, and Andrew Sumner. Torcs, the open racing car simulator. 2005. 2.1

[13] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, pages 1587–1596. PMLR, 2018. 3.1.2

[14] Christine M Hoehner, Carolyn E Barlow, Peg Allen, and Mario Schootman. Commuting distance, cardiorespiratory fitness, and metabolic risk. *American journal of preventive medicine*, 42(6):571–578, 2012. 1.1

[15] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. *Nature*, pages 1–11, 2021. 1.1

[16] Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, and Sergey Levine. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. *CoRR*, abs/1806.10293, 2018. URL http://arxiv.org/abs/1806.10293. 4.1.1

[17] Alex Kendall, Jeffrey Hawke, David Janz, Przemyslaw Mazur, Daniele Reda, John-Mark Allen, Vinh-Dieu Lam, Alex Bewley, and Amar Shah. Learning to drive in a day. *CoRR*, abs/1807.00412, 2018. URL http://arxiv.org/abs/1807.00412. 3.1.2

[18] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. *CoRR*, abs/2006.04779, 2020. URL https://arxiv.org/abs/2006.04779. 4.1.1

[19] Sergey Levine, Peter Pastor, Alex Krizhevsky, and Deirdre Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *CoRR*, abs/1603.02199, 2016. URL http://arxiv.org/abs/1603.02199. 4.1.1

[20] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *CoRR*, abs/2005.01643, 2020. URL https://arxiv.org/abs/2005.01643. 4.1.1

[21] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom

Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015. 3.1.2

[22] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013. 3.1.2

[23] Wassim G Najm, John D Smith, Mikio Yanagisawa, et al. Pre-crash scenario typology for crash avoidance research. Technical report, United States. National Highway Traffic Safety Administration, 2007. 1.1, 2.3.2

[24] OpenAI, Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, Jonas Schneider, Nikolas Tezak, Jerry Tworek, Peter Welinder, Lilian Weng, Qiming Yuan, Wojciech Zaremba, and Lei Zhang. Solving rubik's cube with a robot hand. *CoRR*, abs/1910.07113, 2019. URL http://arxiv.org/abs/1910.07113. 1.1

[25] Lerrel Pinto and Abhinav Gupta. Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours. *CoRR*, abs/1509.06825, 2015. URL http://arxiv.org/abs/1509.06825. 4.1.1

[26] Dean A Pomerleau. Alvinn: An autonomous land vehicle in a neural network. In *Advances in neural information processing systems*, pages 305–313, 1989. 3.1.1

[27] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy P. Lillicrap, and David Silver. Mastering atari, go, chess and shogi by planning with a learned model. *CoRR*, abs/1911.08265, 2019. URL http://arxiv.org/abs/1911.08265. 4.1

[28] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015. 3.1.2

[29] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL http://arxiv.org/abs/1707.06347. 3.1.2

[30] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy P. Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *CoRR*, abs/1712.01815, 2017. URL http://arxiv.org/abs/1712.01815. 1.1, 4.1

[31] Santokh Singh. Critical reasons for crashes investigated in the national motor vehicle crash causation survey, 2018. 1.1

[32] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018. 3.1.2

[33] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063, 2000. 3.1.2

[34] Yang Yanqun, Xinyi Zheng, Jiang Sudan, and Xu Yifan. Driver's reaction time towards red-light timing at urban intersections. *Procedia - Social and Behavioral Sciences*, 96:2506–2510, 11 2013. doi: 10.1016/j.sbspro.2013.08.280. 3.3.1