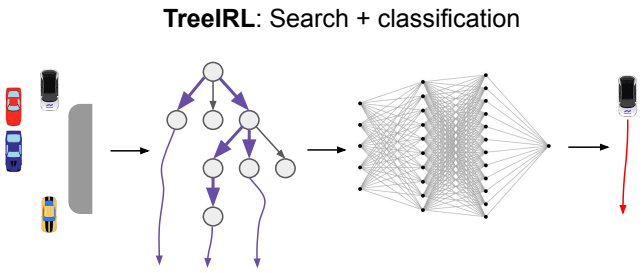
2026 IEEE International Conference on Robotics and Automation (ICRA 2026)
June 1-5, 2026. Vienna, Austria

TreeIRL: Safe Urban Driving with Tree Search and Inverse Reinforcement Learning

Momchil S. Tomov, Sang Uk Lee, Hansford Hendargo, Jinwook Huh, Teawon Han, Forbes Howington, Rafael da Silva, Gianmarco Bernasconi, Marc Heim, Samuel Findler, Xiaonan Ji, Alexander Boule, Michael Napoli, Kuo Chen, Jesse Miller, Boaz Floor, Yunqing Hu
Motional AD Inc.
{momchil.tomov,hansford.hendargo,jinwook.huh,teawon.han,boaz.floor}@motional.com

***Abstract*—We present TreeIRL, a novel planner for autonomous driving that combines Monte Carlo tree search (MCTS) and inverse reinforcement learning (IRL) to achieve state-of-the-art performance in simulation and in real-world driving. The key idea is to use MCTS to find a promising set of safe candidate trajectories and a deep scoring function trained with IRL to select the most human-like among them. We evaluate TreeIRL against classical and state-of-the-art planners on large-scale simulations and on 500+ miles of real-world autonomous driving in the Las Vegas metropolitan area. Scenarios include navigating heavy urban traffic, adaptive cruise control, cut-ins, and traffic lights. TreeIRL achieves the best overall performance, striking a balance between safety, progress, comfort, and human-likeness. To the best of our knowledge, our work is the first public-road demonstration of MCTS-based planning and underscores the importance of evaluating planners across a diverse set of metrics and in real-world environments. TreeIRL is highly extensible and could be further improved with reinforcement learning and imitation learning, providing a framework for exploring different combinations of classical and learning-based approaches to solve the planning bottleneck in autonomous driving.**



	Engineered rules		Learned rules			Hybrid	
	IDM	MCTS	PBP	DriveIRL	Gigaflow	DriveIRL-Safe	TreeIRL
Scales with data	✗	✗	✓	✓	✓	✓	✓
Safe	✗	✓	✗	✗	✓	✓	✓
Comfortable	✗	✗	✗	✓	✗	✗	✓
Human-like	✗	✗	✓	✓	✗	✓	✓
Flexible	✗	✓	✗	✗	✗	✗	✓

Fig. 1. TreeIRL schematic and results summary.

I. Introduction

Human-level planning and decision making remain the holy grail of autonomous driving, promising to make transportation safer, cheaper, and more accessible to everyone. Mirroring broader trends in artificial intelligence, classical approaches to motion planning that explicitly codify the rules of driving in symbolic form [1], [2] have given way to approaches based on machine learning (ML) that learn the rules of driving from data and represent them implicitly in sub-symbolic form [3], [4]. While similar developments have fueled remarkable success in other domains – such as image processing [5], language processing [6], game play [7], and even perception and prediction for self-driving cars [8]–[12] – ML-based planners have struggled to live up to their promise, raising questions about their utility [13].

Some classical approaches frame the planning problem as tree search over an appropriate state space [2], [14]–[16]. These approaches can ensure safety with explicit checks and can flexibly find solutions in a wide variety of on-road situations [14]. However, they can occasionally produce uncomfortable and unnatural behavior, something difficult to remedy with data due to their non-differentiability and small number of parameters.

ML-based approaches often frame the planning problem as trajectory regression or classification and are trained on many hours of human driving [17]–[21]. These approaches scale with data, producing increasingly human-like behavior as their many parameters are refined with training. However, they can struggle to ensure safety, as safety-critical cases are rarely encountered on the road and in the training data [22]. Similarly, they can struggle to generalize flexibly to scenarios outside of the data distribution [3].

In this work, we propose TreeIRL, a hybrid approach that combines classical tree search with a learned classifier to yield a planner that is safe, comfortable, and human-like (Fig. 1). The key insight is to repurpose *Monte Carlo tree search (MCTS) as a trajectory generator*: instead of computing a single next action (e.g., control command), as is usually done, MCTS generates a set of possible future action sequences (i.e., trajectories). These candidate trajectories are fed into a scoring function trained on expert human driving using inverse reinforcement learning (IRL).

See https://arxiv.org/abs/2509.13579 for earlier, longer version of paper.

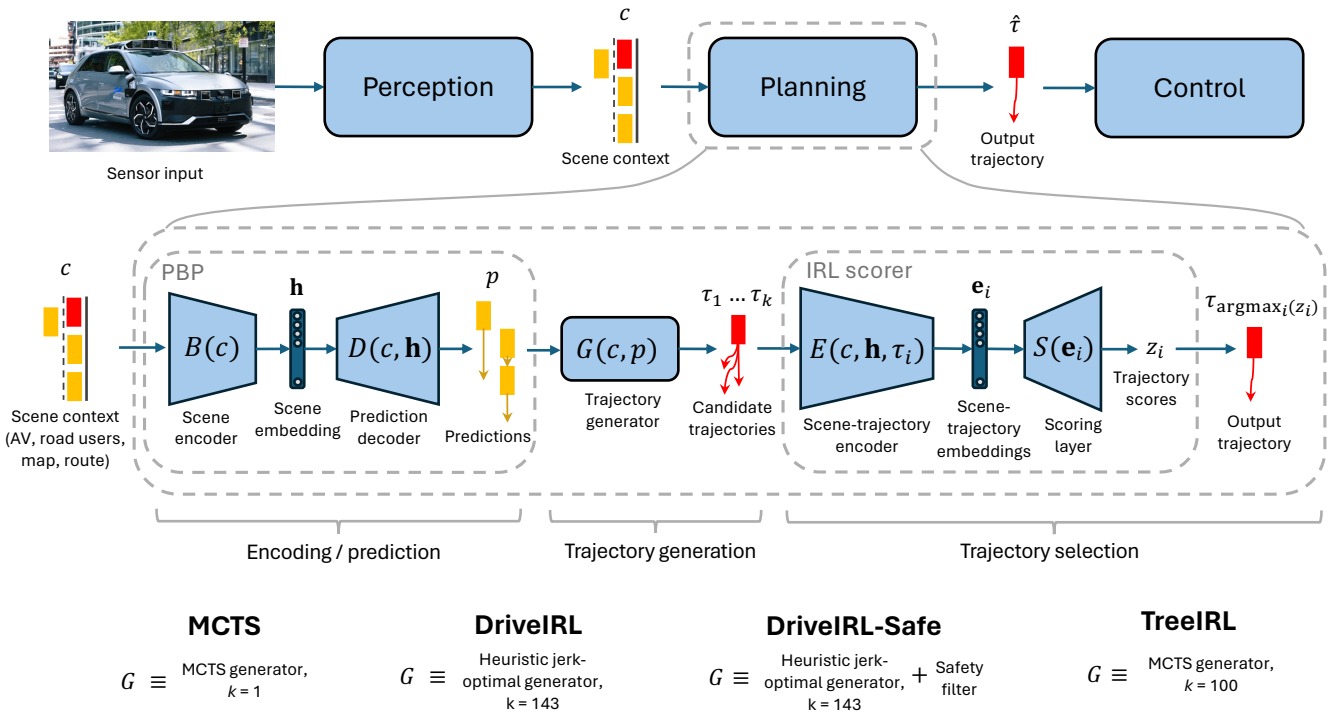


Fig. 2. Planner architecture (skip connections omitted for clarity).

MCTS efficiently explores the trajectory space at decision time to home in on trajectories that are generally appropriate for the current situation, effectively selecting the behavior mode (e.g., slow down, accelerate, stop). The IRL scorer then takes advantage of the variability around that mode to further refine behavior by selecting the most human-like trajectory for execution (e.g., slow down gradually vs. abruptly). Intuitively, this division of labor delegates safety and progress to MCTS and comfort to IRL, although we find empirically that the two complement each other to yield a system that is greater than the sum of its parts.

We evaluate TreeIRL against classical and state-of-the-art motion planners on challenging urban driving scenarios in simulation and in the real world. We find that TreeIRL strikes a balance between safety, comfort, and progress that achieves the best overall performance. Our contributions are as follows: 1) we show how MCTS can be repurposed as a trajectory generator and combined with IRL reap the benefits of both classical and ML-based motion planning, 2) we provide the first real-world demonstration of motion planning based on MCTS in dense urban traffic, 3) we demonstrate the superiority of TreeIRL by comparing it against vanilla MCTS and the state of the art in large-scale evaluations both in simulation and in real-world urban driving, 4) we highlight the need for considering a diverse set of metrics and for performing on-road evaluations to address the simulation-to-reality (sim-to-real) gap.

II. TreeIRL

Planner architecture. We use a version of the DriveIRL architecture (Fig. 2), which uses a modular deep neural network that separates planning into three sub-modules:

- Encoding/prediction: a scene encoder B and decoder D compute a scene embedding $\mathbf{h}$ and predictions p for the other agents. For this module, we use PBP [23].
- Trajectory generation: a trajectory generator G computes a set of k candidate trajectories $\{\tau_1 \dots \tau_k\}$. For DriveIRL, G is a heuristic generator that computes jerk-optimal trajectories that reach a handpicked set of longitudinal target offsets. It is optionally followed by safety filter that excludes trajectories likely to result in collision, a version that we refer to as DriveIRL-Safe.
- Trajectory selection: an IRL scorer, consisting of a scene-trajectory transformer E and a score transformer S, computes a score z_i for each trajectory τ_i, which quantifies how human-like it is.

We pre-train PBP (B and D) for multi-agent motion prediction, as described previously [23]. We then keep those weights frozen and train the IRL scorer (E and S; G is not differentiable) on 1,360 hours of human expert driving using maximum-entropy IRL [21], [24], [25], formulated as a classification problem and optimized using a focal loss [26]. During inference, the trajectory with highest score is selected on each iteration.

The key innovation behind TreeIRL is to replace G with a trajectory generator based on MCTS, which focuses trajectory selection on the narrow part of the trajectory space that is behaviorally appropriate for the current scenario (Fig. 3). For the rest of this section, we describe the technical details of the MCTS trajectory generator.

A. MDP components

We focus on lane following and adaptive cruise control (ACC). The state and action spaces are restricted to 1-D

longitudinal control along a predefined reference path, in our case corresponding to the lane centerline. As we show in the results section, even this simple setup can pose challenges for state-of-the-art approaches.

State space. Each state includes the longitudinal offset x, velocity $\dot{x}$, and acceleration $\ddot{x}$ of the ego vehicle and the lead agent (if there is one), as well as the time offset t in the planning horizon (useful for determining termination and indexing into predictions). It also includes static information, such as the maximum longitudinal offset x_{max} (e.g., the goal, or a red traffic light) and the speed limit $\dot{x}_{\text{max}}$: $s = (x_{\text{ego}}, \dot{x}_{\text{ego}}, \ddot{x}_{\text{ego}}, x_{\text{lead}}, \dot{x}_{\text{lead}}, \ddot{x}_{\text{lead}}, t, x_{\text{max}}, \dot{x}_{\text{max}})$. We set $t = 0$ for the initial state s_0.

Action space. The action corresponds to the longitudinal jerk command, discretized into 5 possibilities: $a = \dddot{x}_{\text{ego}} \in \{-2, -1, 0, 1, 2\}$ m/s^3.

Transition function. The ego and lead agent portions of the next state $s' \sim T(\cdot \mid s, a)$ are computed separately. For the ego, we use a simple kinematic model that forward integrates the jerk command to produce the next ego state with a time step of $\Delta t = 0.5$ s. For the other agents, we use a non-reactive world model based on the (top mode) predictions p from PBP. In particular, the lead agent at time t' is defined as the closest agent projected onto the reference path in front of the ego predicted to be within 2 m of the reference path at time t'. If there is no such agent, the lead portion of the state is left empty. While this results in a simplified non-reactive simulation, it significantly reduces latency compared to a reactive simulation. All transitions are deterministic, so for convenience we can write $s' \leftarrow T(s, a)$.

Reward function. The reward is weighted sum of comfort, tracking, safety, and clearance terms, together with a buffer reward when the ego vehicle stops at a desired distance:

$$
\begin{aligned}
&R(s, a, s') = -\alpha\, cost(s'), \text{ where} \\
&cost(s) \\
&= w_{\text{jerk}}\, \dddot{x}^2_{\text{ego}} && (1)\\
&+ w_{\text{accel}}\, \ddot{x}^2_{\text{ego}} && (2)\\
&+ w_{\text{speed}}\, |\dot{x}_{\text{max}} - \dot{x}_{\text{ego}}| && (3)\\
&- 2\, w_{\text{speed}}\, \mathbb{I}\big[|\dot{x}_{\text{max}} - \dot{x}_{\text{ego}}| < 0.5 \text{ m/s}\big] && (4)\\
&+ w_{\text{collision}}\, \mathbb{I}\big[x_{\text{ego}} \geq x_{\text{lead}}\big]\, (\dot{x}_{\text{lead}} - \dot{x}_{\text{ego}})^2 && (5)\\
&+ w_{\text{collision}}\, \mathbb{I}\big[x_{\text{ego}} \geq x_{\text{max}}\big]\, \dot{x}^2_{\text{ego}} && (6)\\
&+ w_{\text{clearance}}\, \mathbb{I}\big[0 < x_{\text{lead}} - x_{\text{ego}} < \delta\big]\, (x_{\text{lead}} - x_{\text{ego}} - \delta)^2 && (7)\\
&+ w_{\text{clearance}}\, \mathbb{I}\big[0 < x_{\text{max}} - x_{\text{ego}} < \delta\big]\, (x_{\text{max}} - x_{\text{ego}})^2 && (8)\\
&+ w_{\text{stop}}\, \mathbb{I}\big[\dot{x}_{\text{ego}} \approx 0 \wedge (\delta \leq x_{\text{lead}} - x_{\text{ego}} < 3\,\text{m})\big]\, (\dot{x}_{\text{max}} - 2\dot{x}_{\text{ego}}) && (9)\\
&+ w_{\text{stop}}\, \mathbb{I}\big[\dot{x}_{\text{ego}} \approx 0 \wedge (0 \leq x_{\text{max}} - x_{\text{ego}} < 2\,\text{m})\big]\, (\dot{x}_{\text{max}} - 2\dot{x}_{\text{ego}}), && (10)
\end{aligned}
$$

where $\alpha = 1/30$ is a scaling factor and δ is the clearance buffer, which we set to $\delta = 1$ m during training and $\delta = 2$ m during evaluation. Eq. 1 and 2 encourage comfort. Eq. 3 and 4 encourage (roughly) following the speed limit. Eq. 5 and 6 penalize collisions. Eq. 7 and 8 encourage maintaining a certain clearance. Finally, Eq. 9 and 10 encourage the ego not to stop too far behind. If there is no lead agent, the terms involving the lead are set to 0. We use the following reward weights: $w_{\text{jerk}} = 0.05$, $w_{\text{accel}} = 0.2$, $w_{\text{speed}} = 0.1$, $w_{\text{collision}} = 10.0$, $w_{\text{clearance}} = 10.0$, $w_{\text{stop}} = 0.1$. The discount factor is $\gamma = 0.99$.

Termination function. The episode terminates when the planning horizon H is reached. We use $H = 8$ s, which at $\Delta t = 0.5$ s corresponds to maximum tree depth of 16.

B. MCTS components

We use a variant of MCTS based on the AlphaGo algorithm [27], [28] that can incorporate ML to guide the tree search towards more promising actions and thus drastically reduce the sample complexity. In particular, a neural network f_θ parametrized by θ can take a state s and action a and output a policy π_θ and an approximate value estimate $\hat{V}_\theta$: $\big(\pi_\theta(a \mid s), \hat{V}_\theta(s)\big) = f_\theta(s, a)$.

Here, the parameters θ are learned using RL and/or IL. In our case, we use f_θ in the MCTS trajectory generator in three ways: 1) as a prior P in the tree policy to guide selection, 2) as a rollout policy π_{rollout} and/or value function approximator $\hat{V}$ for leaf evaluation, and 3) as a padding policy π_{padding} to generate trajectories from the top k leaves in the end.

Selection. For the tree policy, we use the PUCTS formula [27], which is an extension of the standard upper-confidence bound (UCB) algorithm [29] that uses the tree statistics Q and N to balance exploration – selecting unfamiliar actions with low $N(s, a)$ – and exploitation – selecting rewarding actions with high $Q(s, a)$:

$UCB(s, a) = \frac{Q(s,a)}{Q_{\max}} + c_{\text{puct}}\, P(s, a) \sqrt{\frac{\sum_b N(s,b)+1}{N(s,a)+1}} + \varepsilon$, where $c_{\text{puct}} = 1$ is the UCB scaling factor, with higher values promoting more exploration, $Q_{\max} = 1$ is a normalization factor for the action-value estimates, and $\varepsilon \sim U(0, 0.001)$ is a small amount of noise added for breaking ties. When traversing the tree, actions are selected according to $a = \arg\max_a UCB(s, a)$.

In PUCTS, the prior policy P guides the search by putting greater weight on actions that are *a priori* promising – that is, before simulating their outcomes. In our experiments, we consider two possibilities for the prior policy P: 1) the learned RL policy π_θ, i.e. $P(s, a) = \pi_\theta(a \mid s)$, or 2) a uniform policy $\mathcal{U}_\mathcal{A}$, i.e. $P(s, a) = 1/|\mathcal{A}|$.

Evaluation. There are broadly two ways to estimate the value of a newly visited state s. One option is to use a bootstrap estimate, i.e. the value learned using Bellman updates during training. In our case, this corresponds to the approximate value estimate from the neural network f_θ: $\texttt{EVALUATE}(s) = \hat{V}_\theta(s)$. Another option is to use a Monte Carlo estimate, i.e. the return from a simulated rollout when following policy π_{rollout}: $\texttt{EVALUATE}(s) = \sum_{j=0}^{\infty} \gamma^j\, R(s_j, a_j, s_{j+1})\, \mathbb{I}[F(s_j) = 0]$, where $a_j \sim \pi_{\text{rollout}}(\cdot \mid s_j)$, $s_{j=0} = s$.

We consider three alternatives for π_{rollout}: 1) the learned RL policy, i.e. $\pi_{\text{rollout}} \equiv \pi_\theta$, 2) an IDM policy, i.e. $\pi_{\text{rollout}} \equiv \pi_{\text{IDM}}$, which deterministically chooses the acceleration that an IDM would chose, and 3) a constant speed (CS) policy, i.e. $\pi_{\text{rollout}} \equiv \pi_{\text{CS}}$, which always chooses acceleration 0 m/s^2.

Notice that π_{IDM} and π_{CS} have a different action space (acceleration, $\ddot{x}_{\text{ego}}$ rather than jerk, $\dddot{x}_{\text{ego}}$). Since they are only

used for rollouts, we simply combine them with corresponding transition functions that integrate acceleration instead of jerk; the state space remains unchanged.

Our full MCTS algorithm is shown in Algorithm 1. Note that tree expansion is implicit, as a new node is added for s as soon as $N(s,a) > 0$.

Padding. In order to generate trajectories from the resulting tree, we perform depth-first search (DFS) from the root state s_0 by visiting child nodes in decreasing order of $N(s,a)$ – i.e., most popular children first. Let $\{l_1, l_2 ... l_k\}$ correspond to the first k leaves visited by the DFS (the "top k" leaves). We then perform a rollout from each l_i by following a padding policy π_{padding} until we reach a terminal state. The resulting sequence of state-action pairs from the root stat s_0 to a terminal state corresponds to trajectory τ_i.

We experiment with three possibilities for the padding policy π_{padding} to generate trajectories from the top k leaves of the resulting tree: 1) the learned RL policy, i.e. $\pi_{\text{padding}} \equiv \pi_\theta$, 2) the IDM policy, i.e. $\pi_{\text{padding}} \equiv \pi_{\text{IDM}}$, and 3) the constant speed policy, i.e. $\pi_{\text{padding}} \equiv \pi_{\text{CS}}$.

C. State initialization

The MCTS generator constructs the initial state s_0 at $t = 0$ for the tree search from the following components of the scene context c and predictions p:

- kinematic *ego state*: ego center, orientation, velocity, acceleration, and size (length, width) at the present time in Cartesian baselink 2D frame (origin is the rear axle center, x-direction is forward, y-direction is left),
- kinematic *agent states*: same information for the other agents from the perception module,
- agent *predictions*: predicted trajectories (top mode only) for the other agents from the prediction module as 8-s waypoint sequences at 2 Hz,
- *reference path*: the reference path (in our case, the lane centerline) from the map and routing modules as a sequence of Cartesian 2D segments.

The longitudinal components of the *ego state* projected onto the reference path are used to construct the ego portion of s_0 ($x_{\text{ego}}, \dot{x}_{\text{ego}}, \ddot{x}_{\text{ego}}$). The longitudinal components of the lead agent portion of s_0 ($x_{\text{lead}}, \dot{x}_{\text{lead}}, \ddot{x}_{\text{lead}}$) are determined in the same way as for $t > 0$ (see Transition section), except using the *agent states* instead of the *predictions*. The maximum longitudinal offset x_{max} is either the goal pose or the closest red or yellow traffic light at which the ego can safely stop (whichever is closest). The speed limit $\dot{x}_{\text{max}}$ comes from the map.

D. Trajectory post-processing

The 1-D longitudinal trajectories resulting from the MCTS generator are converted to sequences of Cartesian 2D waypoints by taking the corresponding points along the reference path. These 2-D trajectories are then passed to the IRL scorer. Finally, the top trajectory chosen by the IRL scorer is passed to the downstream post-processing system for smoothing and to ensure kinematic feasibility.

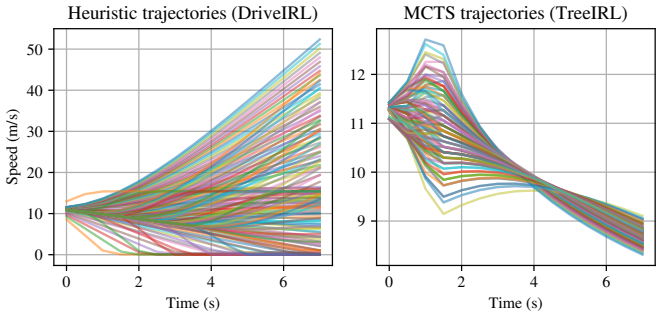


Fig. 3. Example open-loop trajectories for the same scenario.

E. RL network and training

The RL network f_θ is a multilayer perceptron with two hidden layers of 256 units each for both the policy and the value function. The network is trained for a total of 40 million steps using Proximal Policy Optimization (PPO) [30] within Stable-baselines3 [31] using a custom vehicle-following environment. The MDP of the RL agent is the same as the MDP used in MCTS.

Algorithm 1 Monte Carlo tree search (MCTS) algorithm

function SEARCH(s)
  if $F(s)$ **then** ▷ Termination check
    return 0
  end if
  $a \leftarrow \arg\max_a UCB(s,a)$ ▷ Selection
  $s' \leftarrow T(s,a)$ ▷ Transition
  if $N(s,a) = 0$ **then** ▷ Evaluation
    $v \leftarrow \begin{cases} 0 & \text{if } F(s') = 1 \\ \text{EVALUATE}(s') & \text{otherwise} \end{cases}$
  else
    $v \leftarrow \text{SEARCH}(s')$ ▷ Recursive search
  end if
  $r \leftarrow R(s,a,s')$ ▷ Reward
  $q \leftarrow r + \gamma\, v$ ▷ Backup
  $N(s,a) \leftarrow N(s,a) + 1$
  $Q(s,a) \leftarrow Q(s,a) + \dfrac{q - Q(s,a)}{N(s,a)}$
  return q
end function

TABLE I
MCTS LATENCY MEASUREMENTS

n	k	P	$\hat{V}/\pi_{\text{rollout}}$	π_{padding}	Latency (ms)
400	100	π_θ	π_θ	π_θ	911.79 ± 15.26
400	100	π_θ	$\hat{V}_\theta$	π_θ	255.95 ± 4.97
400	100	π_θ	π_{IDM}	π_{IDM}	45.67 ± 1.14
400	100	π_θ	π_{CS}	π_{IDM}	52.32 ± 2.69
400	100	$\mathcal{U}_\mathcal{A}$	π_θ	π_θ	691.72 ± 8.73
400	100	$\mathcal{U}_\mathcal{A}$	π_θ	π_{IDM}	569.49 ± 8.49
400	100	$\mathcal{U}_\mathcal{A}$	$\hat{V}_\theta$	π_θ	181.38 ± 2.73
400	100	$\mathcal{U}_\mathcal{A}$	$\hat{V}_\theta$	π_{IDM}	48.66 ± 0.65
400	100	$\mathcal{U}_\mathcal{A}$	π_{IDM}	π_θ	151.42 ± 4.03
400	100	$\mathcal{U}_\mathcal{A}$	π_{IDM}	π_{IDM}	10.05 ± 2.79
400	100	$\mathcal{U}_\mathcal{A}$	π_{IDM}	π_{CS}	6.00 ± 0.14
400	100	$\mathcal{U}_\mathcal{A}$	π_{CS}	π_{IDM}	4.88 ± 0.10
400	100	$\mathcal{U}_\mathcal{A}$	π_{CS}	π_{CS}	4.93 ± 0.14
0	1	n/a	n/a	π_θ	2.21 ± 0.05
0	1	n/a	n/a	π_{IDM}	0.03 ± 0.001

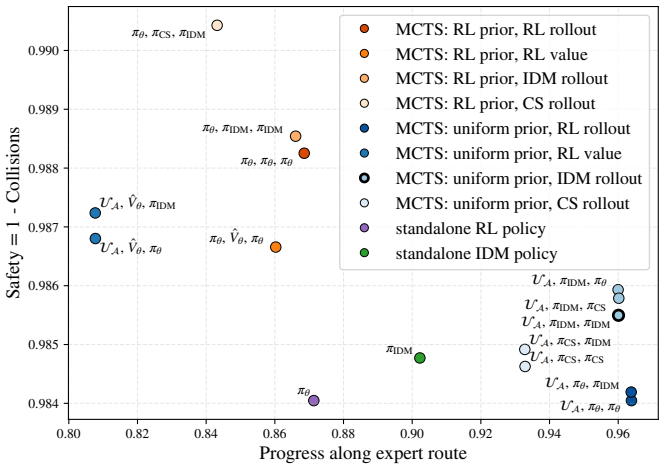


Fig. 4. Comparing MCTS configurations.

III. NuPlan Experiments

We first compare different configurations of MCTS (Fig. 4). Given the latency costs (Tab. I) of the learned policy π_θ and its overly conservative bias, we use the following parameters for the MCTS generator in subsequent experiments: $n = 400, k = 100, P \equiv \mathcal{U}_\mathcal{A}, \pi_{\text{rollout}} \equiv \pi_{\text{IDM}}, \pi_{\text{padding}} \equiv \pi_{\text{IDM}}$.

We then evaluate TreeIRL in nuPlan [4] against classical and state-of-the-art planners on scenarios based on real-world driving logs using the nuPlan simulator [4]. Our focus is on lane following and adaptive cruise control (ACC). As we show, even this restricted domain poses challenges to state-of-the-art approaches. We consider the following baselines:

- The intelligent driver model (**IDM**) [32]: a classical planner that computes the acceleration for collision-free ACC in closed form,
- **MCTS**, corresponding to TreeIRL with $k = 1$ trajectory, i.e. treating the IRL scorer as pass-through,
- Path-based prediction (**PBP**) [23]: an open-loop motion forecasting model trained using imitation learning,
- **DriveIRL** [21]: precursor to TreeIRL using the same planner architecture, except with a heuristic trajectory generator that generates jerk-optimal trajectories reaching a set of predefined target points,
- **Gigaflow** [33]: a RL-based motion planner trained in closed loop using self-play, with the same policy controlling the ego and all other agents,
- **DriveIRL-Safe** [21]: identical to DriveIRL, with the addition of a safety filter to exclude any apparently unsafe trajectories.

Most planners exhibit comparable and acceptable safety and progress (Tab. II), with the exception of IDM, PBP (collisions), and Gigaflow (drivable area violations). Comfort is comparable across planners, except for Gigaflow. IDM comfort is also on the lower side, especially longitudinal jerk and acceleration. Similarity to the human expert driver is comparable across planners, except for Gigaflow and IDM. Based on these results, we exclude IDM, PBP, and Gigaflow from subsequent experiments and analyses, focusing only on MCTS, DriveIRL, DriveIRL-Safe, and TreeIRL.

IV. Real-world driving

The ultimate test for a planner is how it performs in the real world. To that end, we deploy the four most promising planners on Hyundai IONIQ 5 self-driving cars and evaluate them on public roads in Las Vegas [16] (Tab. III). As in previous work [16], prior to on-road deployment, performance of the entire AV stack is thoroughly evaluated using the Object Sim simulator from Applied Intuition [34], which performs realistic high-fidelity physics simulation of vehicle dynamics.

Simulations largely recapitulate the nuPlan results, with DriveIRL showing better comfort and TreeIRL showing better safety. TreeIRL generally shows improved comfort over MCTS while having relatively comparable safety and progress, highlighting the benefit of the IRL scorer. DriveIRL-Safe has comparable safety to TreeIRL (except for traffic light and stop line violations), but worse comfort overall. Analyses of individual simulations (Fig. 5) show that DriveIRL fails to stop on time to prevent collisions, while DriveIRL-Safe stops too soon and too abruptly.

The on-road results, however, overwhelmingly favor TreeIRL across all metrics. Safety is $\sim$2 orders of magnitude better than DriveIRL and 2-4x better than DriveIRL-Safe and MCTS, with zero takeovers by the safety driver due to ACC or cut-in failures across all 268 auto-miles. Progress is comparable to MCTS and substantially better than DriveIRL and DriveIRL-Safe. Analysis of individual on-road cut-ins (Fig. 6) reveal similar kinematic profiles to the simulations: MCTS deceleration is somewhat jerky, DriveIRL slows down insufficiently to prevent takeovers, DriveIRL-Safe brakes too aggressively, and TreeIRL slows down with the smoothest deceleration profile. Interestingly, in contrast to the simulations, both subjective and objective comfort is better for TreeIRL compared to DriveIRL.

V. Conclusion

The substantial advantage of TreeIRL on the road compared to simulation is partly due to the subjective nature of the discretionary metrics. Specifically, the safety driver may take over if AV behavior is perceived to be unsafe, even if it would not have resulted in a collision. Indeed, resimulations confirm that this is the case for a substantial number of takeovers. However, as a measure of perceived safety, takeovers better reflect the subjective experience of the passenger, which in some sense is the ultimate yardstick for measuring AV performance. The sudden braking during takeovers additionally impacts comfort, which is likely the main factor behind the worse comfort of DriveIRL on the road compared to the simulations. Overall, the striking sim-to-real gap [35], [36] in our work highlights the importance of real-world evaluations across a broad set of metrics.

An alternative way to combine MCTS with IRL would be to learn the reward function R (Eq. 1-10) from data. This could obviate the need for the learned trajectory scorer S, as the reward function itself would already capture human-likeness. However, a deep learned reward in the MCTS loop may significantly increase demands on inference compute.

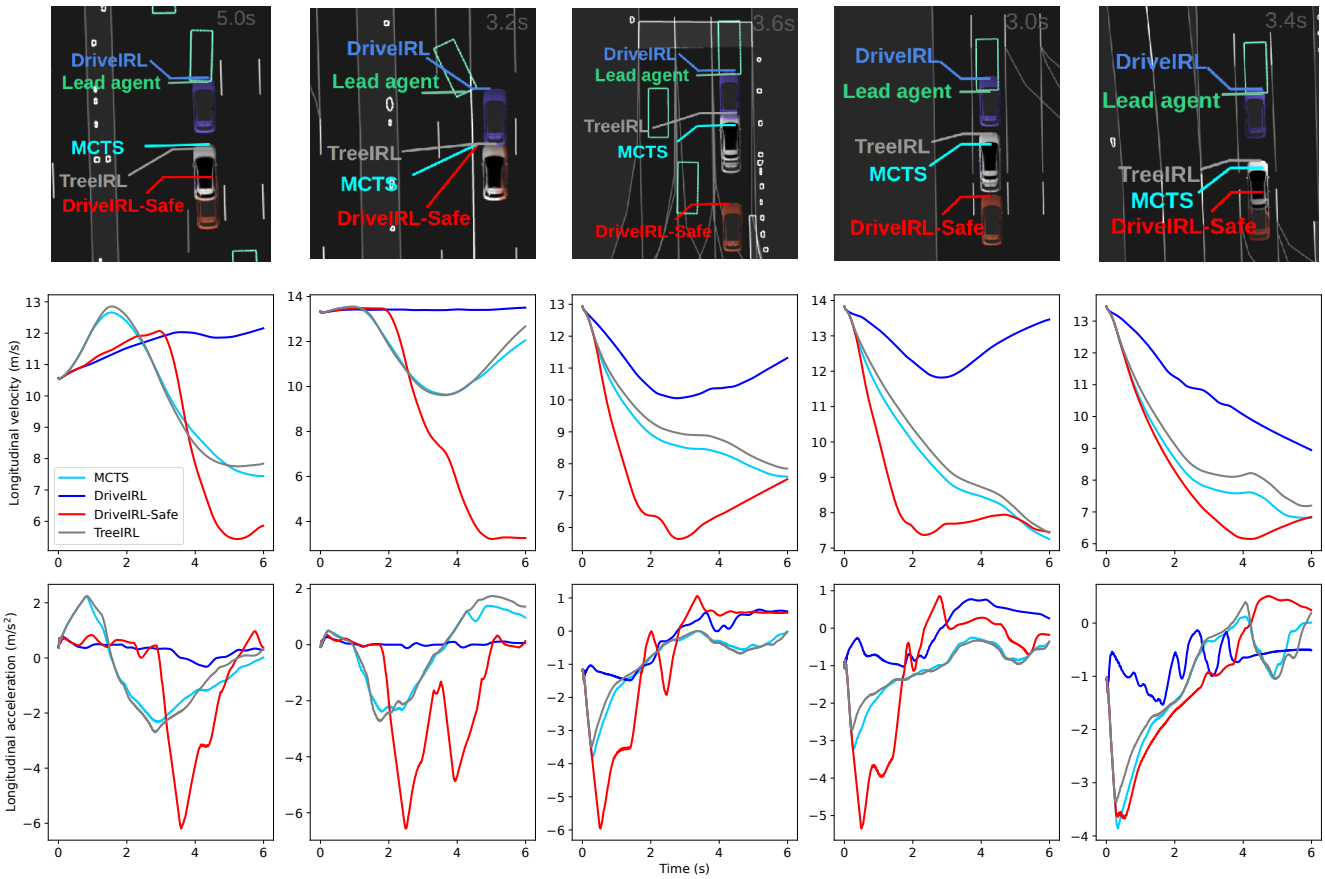


Fig. 5. High-fidelity simulation examples.

Fig. 6. Cut-in examples from real-world driving. Inset, recent history of AV kinematics. For DriveIRL, $t = 0$ corresponds to a safety-related takeover (cut-in failure).

TABLE II
NUPLAN EVALUATION (7,051 SCENARIOS)

Category	Metric	IDM	MCTS	PBP	DriveIRL	Gigaflow	DriveIRL-Safe	TreeIRL
Safety	Collisions ↓	0.02	**0.01**	0.05	**0.01**	**0.01**	**0.01**	**0.01**
	Drivable area violations ↓	**0.01**	**0.01**	0.03	**0.01**	0.11	**0.01**	**0.01**
	Traffic light violations ↓	**0.02**	**0.02**	0.05	**0.02**	0.04	**0.02**	**0.02**
	Speed limit violations ↓	**0.09**	0.28	0.18	0.12	0.29	0.12	0.20
	Time gap (s) ↑	3.73	3.68	3.63	3.85	3.42	**3.97**	3.90
Progress	Progress along expert route ↑	0.90	**0.96**	0.92	0.90	0.89	0.89	0.92
Comfort	Comfort ↑	0.92	0.98	0.97	**0.99**	0.38	0.93	0.98
	Min longitudinal jerk (m/s^3) ↑	-1.08	-0.87	-0.66	**-0.44**	-2.59	-0.61	-0.77
	Max longitudinal jerk (m/s^3) ↓	1.06	1.16	0.88	**0.55**	2.58	0.71	0.84
	Min longitudinal accel (m/s^2) ↑	-0.99	-0.67	-0.52	**-0.43**	-1.65	-0.47	-0.59
	Max longitudinal accel (m/s^2) ↓	0.41	0.75	0.67	**0.38**	1.74	0.55	0.57
Human-likeness	L2 error (m) ↓	4.53	3.82	3.86	**3.53**	8.44	3.56	3.75
	Deceleration delay error (s) $\lvert\downarrow\rvert$	0.32	0.31	0.42	**0.14**	0.78	0.15	0.43
	Acceleration delay error (s) $\lvert\downarrow\rvert$	**0.06**	-0.23	-0.56	0.09	-0.41	0.23	-0.11
	Max speed error ↓	0.91	**0.27**	0.45	1.25	0.43	1.17	0.45

TABLE III
HIGH-FIDELITY SIMULATION AND ON-ROAD EVALUATION

Category	Metric	MCTS	DriveIRL	DriveIRL-Safe	TreeIRL
	High-fidelity 30-s simulations				
	Total simulations	717	717	717	717
Safety	Front collisions ↓	0.04	0.05	**0.03**	**0.03**
	Traffic light violations ↓	**0.01**	0.04	0.04	**0.01**
	Stop line violations ↓	**0.01**	0.06	0.04	**0.01**
	Speed limit violation time (s) ↓	3.81	4.39	**2.07**	5.01
	ACC distance violations (< 1.5 m) ↓	**0.17**	0.20	0.24	0.19
	Min time gap (s) ↑	**1.56**	1.29	1.31	1.45
Progress	Average speed (m/s) ↑	**8.88**	8.41	8.31	**8.88**
	Rear collisions ↓	0.17	**0.16**	0.28	0.19
Comfort	Brake taps ↓	0.27	**0.18**	0.32	0.22
	Longitudinal jerk violations ↓	0.23	**0.12**	0.23	0.17
	Longitudinal accel violations ↓	0.49	**0.23**	0.31	0.56
	Public road evaluation				
	Total auto-miles	115.8	64.3	87.9	268.4
	Discretionary metrics				
Safety	Safety takeovers (mi/event) ↑	7.68	1.43	6.76	**17.89**
	ACC failures (mi/event) ↑	57.9	2.57	87.9	>**268.4**
	Traffic light failures (mi/event) ↑	19.3	8.04	12.56	**67.10**
	Cut-in failures (mi/event) ↑	>**115.8**	5.36	43.95	>**268.4**
Progress	Slow driving (mi/event) ↑	>**115.8**	5.85	2.84	**134.2**
Comfort	Discomfort (mi/event) ↑	1.40	1.74	1.05	**2.42**
	Jerky driving (mi/event) ↑	2.83	6.43	2.20	**13.42**
	Harsh discomfort braking (mi/event) ↑	6.09	4.95	3.66	**8.95**
	Mild discomfort braking (mi/event) ↑	2.83	3.22	3.26	**12.78**
	Objective metrics				
Comfort	Harsh discomfort braking (mi/event) ↑	8.91	4.29	4.88	**9.59**
	Mild discomfort braking (mi/event) ↑	7.24	**64.3**	17.58	15.79
	Brake taps (mi/event) ↑	1.08	0.43	1.03	**1.11**

Yet another way to rid of S would be to train f_θ with a hybrid of RL and IL [37].

Our work can be further extended in multiple ways. Prediction uncertainty could be accounted for by considering multiple prediction modes and averaging rewards over them. Alternatively, predictions could be replaced by a reactive world model, such as TrafficSim [38] or Gigaflow [33]. Gigaflow could also be used to replace the RL network f_θ, allowing the expansion of the state/action space to include lateral control, which would likely require prohibitively many iterations if a simple policy such as the IDM is used. Overall, TreeIRL can be seen as a framework for tackling the planning bottleneck in autonomous driving by combining the strengths of tree search, RL, IL, and IRL in a single planner architecture that facilitates robust comparisons in simulation and in the real world.

ACKNOWLEDGMENTS

We thank Raveen Ilaalagan, Ernest Elizalde, and Xavier Escobar for road tests; Michael Zahniser, Curtis Chan, Samee Mahbub, and Joseph Baker for deployment/integration.

REFERENCES

[1] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006.

[2] B. Paden, M. Čáp, S. Z. Yong, D. Yershov, and E. Frazzoli, "A survey of motion planning and control techniques for self-driving urban vehicles," *IEEE Transactions on Intelligent Vehicles*, vol. 1, no. 1, pp. 33–55, 2016.

[3] S. Kuutti, R. Bowden, Y. Jin, P. Barber, and S. Fallah, "A survey of deep learning applications to autonomous vehicle control," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 2, pp. 712–733, 2020.

[4] N. Karnchanachari, D. Geromichalos, K. S. Tan, N. Li, C. Eriksen, S. Yaghoubi, N. Mehdipour, G. Bernasconi, W. K. Fong, Y. Guo, and H. Caesar, "Towards learning-based planning: The nuplan benchmark for real-world autonomous driving," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*, 2024, pp. 629–636.

[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds., vol. 25. Curran Associates, Inc., 2012. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

[6] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[7] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[8] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom, "nuscenes: A multimodal dataset for autonomous driving," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 11 621–11 631.

[9] Y. Zhou and O. Tuzel, "Voxelnet: End-to-end learning for point cloud based 3d object detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018, pp. 4490–4499.

[10] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom, "Pointpillars: Fast encoders for object detection from point clouds," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019, pp. 12 697–12 705.

[11] T. Phan-Minh, E. C. Grigore, F. A. Boulton, O. Beijbom, and E. M. Wolff, "Covernet: Multimodal behavior prediction using trajectory sets," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 14 074–14 083.

[12] T. Yin, X. Zhou, and P. Krahenbuhl, "Center-based 3d object detection and tracking," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021, pp. 11 784–11 793.

[13] D. Dauner, M. Hallgarten, A. Geiger, and K. Chitta, "Parting with misconceptions about learning-based vehicle motion planning," in *Conference on Robot Learning*. PMLR, 2023, pp. 1268–1281.

[14] D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel, "Path planning for autonomous vehicles in unknown semi-structured environments," *The international journal of robotics research*, vol. 29, no. 5, pp. 485–501, 2010.

[15] Z. N. Sunberg, C. J. Ho, and M. J. Kochenderfer, "The value of inferring the internal state of traffic participants for autonomous freeway driving," in *2017 American control conference (ACC)*. IEEE, 2017, pp. 3004–3010.

[16] M. Heim, F. Suárez-Ruiz, I. Bhuiyan, B. Brito, and M. S. Tomov, "Lab2car: A versatile wrapper for deploying experimental planners in complex real-world environments," in *2025 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2025, pp. 14 828–14 834.

[17] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang *et al.*, "End to end learning for self-driving cars," *arXiv preprint arXiv:1604.07316*, 2016.

[18] F. Codevilla, E. Santana, A. M. Lopez, and A. Gaidon, "Exploring the limitations of behavior cloning for autonomous driving," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019, pp. 9329–9338.

[19] M. Bansal, A. Krizhevsky, and A. Ogale, "Chauffeurnet: Learning to drive by imitating the best and synthesizing the worst," in *Proceedings of Robotics: Science and Systems (RSS)*, 2019.

[20] O. Scheel, L. Bergamini, M. Wolczyk, B. Osiński, and P. Ondruska, "Urban driver: Learning to drive from real-world demonstrations using policy gradients," in *Conference on Robot Learning*. PMLR, 2022, pp. 718–728.

[21] T. Phan-Minh, F. Howington, T.-S. Chu, M. S. Tomov, R. E. Beaudoin, S. U. Lee, N. Li, C. Dicle, S. Findler, F. Suarez-Ruiz *et al.*, "Driveirl: Drive in real life with inverse reinforcement learning," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 1544–1550.

[22] W. Zhou, Z. Cao, Y. Xu, N. Deng, X. Liu, K. Jiang, and D. Yang, "Long-tail prediction uncertainty aware trajectory planning for self-driving vehicles," in *2022 IEEE 25th International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2022, pp. 1275–1282.

[23] S. Afshar, N. Deo, A. Bhagat, T. Chakraborty, Y. Shao, B. R. Buddharaju, A. Deshpande, and H. Cui, "Pbp: Path-based trajectory prediction for autonomous driving," *arXiv preprint arXiv:2309.03750*, 2023.

[24] B. D. Ziebart, A. L. Maas, J. A. Bagnell, A. K. Dey *et al.*, "Maximum entropy inverse reinforcement learning." in *Aaai*, vol. 8. Chicago, IL, USA, 2008, pp. 1433–1438.

[25] M. Wulfmeier, P. Ondruska, and I. Posner, "Maximum entropy deep inverse reinforcement learning," *arXiv preprint arXiv:1507.04888*, 2015.

[26] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2980–2988.

[27] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[28] C.-J. Hoel, K. Driggs-Campbell, K. Wolff, L. Laine, and M. J. Kochenderfer, "Combining planning and deep reinforcement learning in tactical decision making for autonomous driving," *IEEE transactions on intelligent vehicles*, vol. 5, no. 2, pp. 294–305, 2019.

[29] P. Auer, "Using confidence bounds for exploitation-exploration trade-offs," *Journal of machine learning research*, vol. 3, no. Nov, pp. 397–422, 2002.

[30] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[31] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021. [Online]. Available: http://jmlr.org/papers/v22/20-1364.html

[32] M. Treiber, A. Hennecke, and D. Helbing, "Congested traffic states in empirical observations and microscopic simulations," *Physical review E*, vol. 62, no. 2, p. 1805, 2000.

[33] M. Cusumano-Towner, D. Hafner, A. Hertzberg, B. Huval, A. Petrenko, E. Vinitsky, E. Wijmans, T. Killian, S. Bowers, O. Sener *et al.*, "Robust autonomy emerges from self-play," *arXiv preprint arXiv:2502.03349*, 2025.

[34] "Applied Intuition Object Sim," https://www.appliedintuition.com/products/object-sim, accessed: 2024-05-12.

[35] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," in *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE, 2017, pp. 23–30.

[36] B. R. Kiran, I. Sobh, V. Talpaert, P. Mannion, A. A. Al Sallab, S. Yogamani, and P. Pérez, "Deep reinforcement learning for autonomous driving: A survey," *IEEE transactions on intelligent transportation systems*, vol. 23, no. 6, pp. 4909–4926, 2021.

[37] Y. Lu, J. Fu, G. Tucker, X. Pan, E. Bronstein, R. Roelofs, B. Sapp, B. White, A. Faust, S. Whiteson *et al.*, "Imitation is not enough: Robustifying imitation with reinforcement learning for challenging driving scenarios," in *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2023, pp. 7553–7560.

[38] S. Suo, S. Regalado, S. Casas, and R. Urtasun, "Trafficsim: Learning to simulate realistic multi-agent behaviors," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 10 400–10 409.