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

Momchil S. Tomov, Sang Uk Lee, Hansford Hendrago, 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,sang.lee,hansford.hendargo,boaz.floor,alex.hu}@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 core idea is to use MCTS to find a promising set of safe candidate trajectories and a deep IRL scoring function to select the most human-like among them. We evaluate TreeIRL against both classical and state-of-the-art planners in large-scale simulations and on 500+ miles of real-world autonomous driving in the Las Vegas metropolitan area. Test scenarios include dense 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 our knowledge, our work is the first demonstration of MCTS-based planning on public roads 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.**

***Index Terms*—Self-driving cars, autonomous driving, motion planning, Monte Carlo tree search, inverse reinforcement learning.**

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]–[15] – ML-based planners have struggled to live up to their promise, raising questions about their utility [16].

Some classical approaches frame the planning problem as tree search over an appropriate state space [2], [17]–[19]. These approaches can ensure safety with explicit checks and can flexibly find solutions in a wide variety of on-road scenarios [17]. 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.

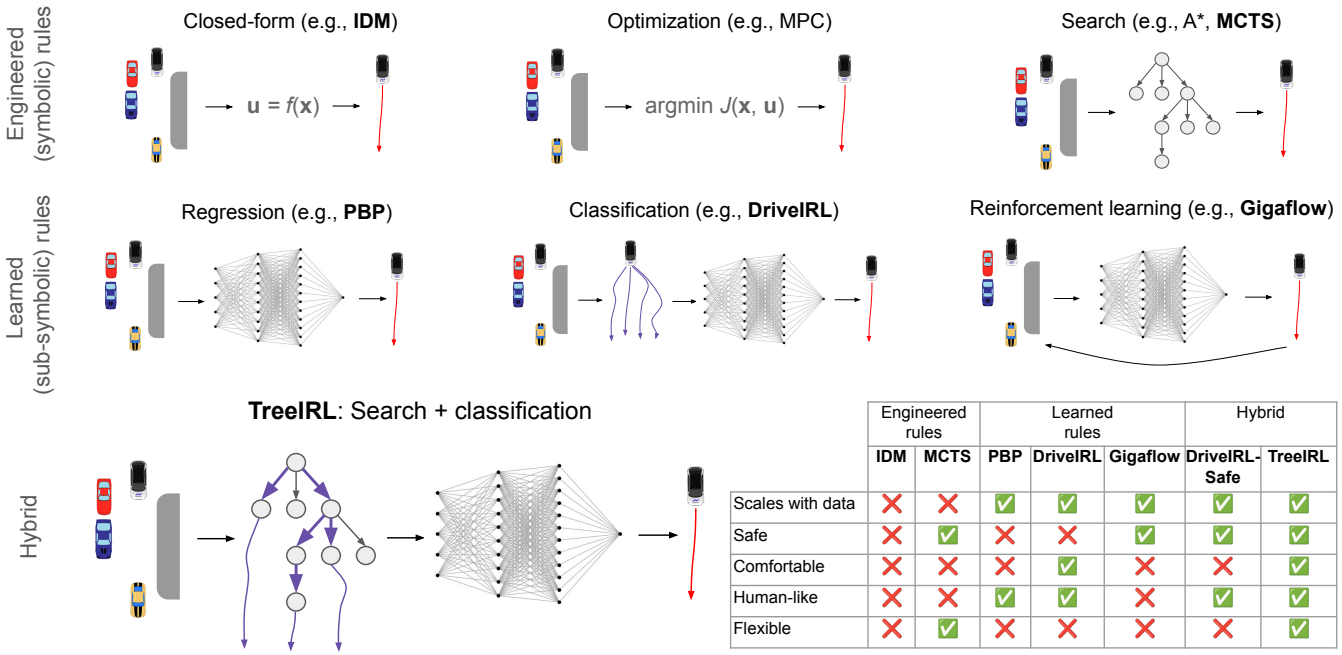
ML-based approaches often frame the planning problem as trajectory regression or classification and are trained on many hours of human driving [20]–[24]. These approaches scale with data, producing increasingly human-like behavior as their many parameters are refined with training. However, they can struggle to ensure safety, as safety-critical cases are rarely encountered on the road and in the training data [25]. 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 main idea is to repurpose *Monte Carlo tree search (MCTS) as a trajectory generator*: instead of computing a single immediate 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).

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 across multiple metrics 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

† – work done while at Motional.



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

Fig. 1. Landscape of motion planning approaches and summary of our results.

as follows:

1) We show how MCTS can be repurposed as a trajectory generator and combined with IRL to reap the benefits of both classical and ML-based motion planning.
2) We show how to optimize the latency of the resulting system in the context of a full autonomous vehicle (AV) stack, making it suitable for real-world deployment without sacrificing driving performance.
3) We provide the first real-world demonstration of motion planning based on MCTS in dense urban traffic.
4) We demonstrate the superiority of TreeIRL by comparing it against vanilla MCTS and other state-of-the-art planners in large-scale evaluations both in simulation and in real-world urban driving.
5) 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. Related Work

Classical approaches to the motion planning problem rely on explicit hand-crafted rules that dictate driving behavior at decision time [1], [2]. The seminal intelligent driver model (IDM) [26] directly computes the command acceleration to maintain a safe distance to a lead vehicle, ensuring collision-free behavior for simple 1-D adaptive cruise control (ACC). Balancing more complex objectives (e.g., reaching a goal while maintaining kinematic visibility and avoiding obstacles) in higher-dimensional environments can be framed as a continuous optimization problem and solved, for example, using model-predictive control (MPC) [27]–[29]. Alternatively, the search space can be discretized and explored using graph search methods such as A* [17], [30] or MCTS [18], [31]. Search and optimization can be combined – for example, A* can find a collision-free path to the goal, while MPC can compute a dynamically feasible trajectory to track it [19].

Classical approaches can produce reasonable driving behavior while providing interpretability and, under certain conditions, safety guarantees. Search methods in particular have the potential to handle a wide variety of scenarios as they can flexibly reason and find solutions even in unusual situations (e.g, navigating a crowded parking lot after a football game). However, handcrafted rules occasionally lead to uncomfortable and unnatural behavior. Furthermore, they have to be tuned manually, which can be time-consuming and scale poorly to scenarios requiring subtly different rules, such as different geolocations or different times of day.

Imitation learning (IL) – or behavior cloning – approaches learn the rules of driving from human demonstrations and store them implicitly in the weights of a neural network [20]–[23]. The promise of IL is scalability: behavior should improve with more training data. Simple regression methods such as path-based trajectory prediction (PBP) have achieved state-of-the-art performance on motion forecasting [32].

With sufficient training, IL can yield comfortable, human-like driving behavior in the majority of scenarios. However, it can struggle on safety-critical edge cases that are underrepresented in the training data (e.g., aggressive cut-ins, jaywalkers) [25]. More generally, naïve IL suffers from a distribution shift problem [3], [33], [34]: the discrepancy between the scenarios observed during training and test time leads to subtle deviations in behavior, which are compounded as errors accumulate in closed loop, leading to situations completely outside of the training distribution and to undefined behavior.

Reinforcement learning (RL) approaches mitigate the distribution shift problem by training a driving policy to

satisfy a reward function using closed-loop simulations [35]–[37]. Through trial-and-error, the policy learns to take actions that lead to reward (e.g., making progress along the route) while avoiding punishment (e.g., colliding with other agents). By virtue of experiencing the outcomes of their own actions and a much wider variety of scenarios than typically countered on the road – including many collisions and the associated "pain" – the resulting policies are more robust to perturbations and produce safer driving than IL.

While safe and scalable, RL approaches depend heavily on the reward function and the simulation environment, which pose their own set of challenges. Designing a reward function that balances different driving objectives in a way that leads to safe, human-like driving carries many of the same difficulties as designing good classical planners [38]. Designing a realistic simulation environment with human-like reactive agents can prove just as difficult as solving the planning problem itself [39], [40]. Training with simple reactive agents can produce unnatural behaviors (e.g., no fear of rear collisions), and so can training with realistic but non-reactive agents replayed from real-world driving data (e.g., excessive fear of rear collisions) [4]. Recently, Gigaflow [40] addressed this circular dependency using self-play: the same policy that controls the AV also controls the other agents, ensuring the simulation also improves gradually over the course of training. This simple concept – combined with massive amounts of training – has allowed Gigaflow to surpass the state of the art on multiple autonomous driving benchmarks.

Inverse reinforcement learning (IRL) promises to address the reward design problem by assuming that human driving is guided by an implicit reward function, which IRL attempts to reverse engineer from human demonstrations [41], [42]. In particular, DriveIRL [24] uses this idea to cast motion planning as a classification problem: a trajectory generator proposes a set of candidate trajectories and a learned reward function selects the best trajectory among them. DriveIRL demonstrated comfortable, human-like driving on a real self-driving car in Las Vegas. However, it inherits some of safety issues of IL, requiring multiple takeovers by the safety driver.

Hybrid methods promise to combine the benefits of different approaches. In DriveIRL, "bad" trajectories proposed by the generator can be excluded using a rule-based safety filter [24], [43], substantially improving safety. As another example, IL can be combined with RL to produce a policy that is human-like and robust in rare, safety-critical scenarios [44]. SafetyNet [45] projects an infeasible ML trajectory onto a heuristically generated set of lane-follow trajectories. Lab2Car [19] uses the ML trajectory to construct a set of spatiotemporal constraints (a "maneuver") that capture comfort and safety, which are then satisfied by MPC.

MCTS offers a particular kind of hybrid that combines classical search with ML in a principled way. The landmark defeat of Go champion Lee Sedol by AlphaGo was achieved with MCTS guided by a policy trained with IL and RL [46]. Subsequent work combining MCTS with ML has achieved similarly groundbreaking results in a number of other domains, including other board games [47] and video games [48]. Recent work has demonstrated that MCTS combined with ML also holds promise in the domain of autonomous driving [49], [50].

The ability of ML-guided MCTS to explore intractably large trajectory spaces at decision time can in principle lead to robust, flexible driving across a diverse set of scenarios. However, the main challenge is latency: the number of iterations necessary to find a good solution grows exponentially with the search space. While ML can significantly reduce that number [49], this comes at the cost of running the ML policy at each iteration of MCTS, which can offset those gains by dramatically increasing overall latency. As a result, to the best of our knowledge, all reported applications of MCTS to self-driving cars have been evaluated only in simulation.

The key insight of our approach is that, instead of producing a single best action or trajectory, *MCTS can produce a set of candidate trajectories*. These trajectories can then be scored by a reward function learned with IRL to choose the one which is most human-like. This relaxes the requirements on MCTS, allowing for a simpler version that fits into the computational budget of a real self-driving car without sacrificing overall driving performance.

III. Theoretical background

In this section, we review the theoretical background of MCTS.

A. Markov Decision Process (MDP)

We build on the MDP formalism developed in the RL literature [51]. A MDP is a tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, T, R, \gamma, F \rangle$ where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $T : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is the transition distribution, $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the reward function, and $\gamma \in (0, 1]$ is the discount factor. The transition distribution $T(s' \mid s, a)$ describes the probability of transitioning to state s' after taking action a in state s. The reward function $R(s, a, s')$ describes the reward obtained by the agent after taking action a in state s and transitioning to state s'. Additionally, it is useful to introduce a termination function $F(s) \to \{0, 1\}$ that denotes whether state s is terminal.

At decision time, the goal of the agent is to choose an action a^* that maximizes its expected return (i.e., sum of future discounted rewards), starting from current state s_0:

$$a^* = \arg\max_{a \in \mathcal{A}} \mathbb{E}\left[\sum_{j=0}^{\infty} \gamma^j R(s_j, a_j, s_{j+1})\right], \qquad (1)$$

where j denotes the time step in the episode and the expectation is taken over the distribution of future states $s_{j+1} \sim T(\cdot \mid s_j, a_j)$ and actions $a_j \sim \pi(\cdot \mid s_j)$ resulting from the transition distribution T and the agent's policy $\pi : \mathcal{S} \to \mathcal{A}$.

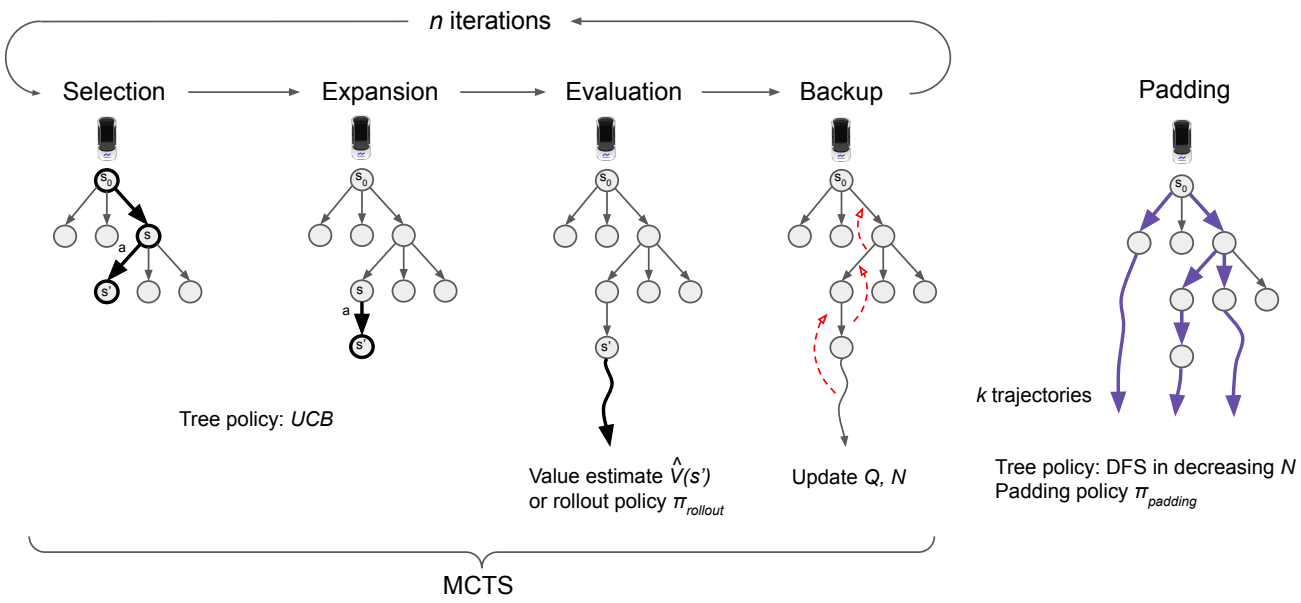


Fig. 2. MCTS trajectory generator.

B. Rollout algorithms

Rollout algorithms correspond to a class of decision-time Monte Carlo planning algorithms that solve Eq. 1 by repeatedly sampling trajectories from the initial state s_0 following a *rollout policy* π_{rollout}, simulating their outcomes, and choosing the action a^* that produces the highest average return across the simulations. These approaches fall under the banner of model-based RL, as they require complete model of the environment – the MDP, or an estimate of it – in order to perform the simulations.

C. Monte Carlo tree search (MCTS)

MCTS improves on rollout algorithms by taking advantage of the fact that most trajectories will share the same initial states, which means that previously sampled trajectories can inform future trajectories (e.g., if going left consistently leads to simulated collisions, we should try something else) [31].

In particular, MCTS incrementally builds a tree rooted in the present state s_0 in which nodes correspond to states and edges correspond to actions leading to next states (child nodes). MCTS expands the tree in multiple iterations, where each iteration consists of the following four steps (Fig. 2) [51]:

1) Selection: starting from the root state s_0, follow a *tree policy* that descends down the tree, balancing exploration and exploitation.
2) Expansion: once a new state is reached, (optionally) add it to the tree as a new leaf node.
3) Evaluation: estimate the expected return from the leaf state, either using a value function approximator $\hat{V}$ (bootstrap estimate), or by computing the simulated return using a rollout policy π_{rollout} (Monte Carlo estimate).
4) Backup: update the tree value estimates $Q(s, a)$ and visit counts $N(s, a)$ for all leaf ancestors.

After n of iterations – often dictated by a compute budget – the tree has $O(n)$ nodes. In the end, the best action can be chosen either as $a^* = \arg\max_a Q(s_0, a)$ or $a^* = \arg\max_a N(s_0, a)$.

D. MCTS as a trajectory generator

The main novel idea behind TreeIRL is that MCTS can be repurposed to generate a set of trajectories, i.e. promising sequences of actions, rather than a single next action a^* (Fig. 2, right). After n iterations, we perform depth-first search (DFS) from the root state s_0 by visiting child nodes in decreasing order of $N(s, a)$ – that is, 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 state s_0 to a terminal state corresponds to trajectory τ_i.

IV. TreeIRL

This section includes a technical description of the TreeIRL planner.

Planner architecture. Our focus is the planning module of the AV stack (Fig. 3, top), which is responsible both for high-level decision making (the behavior mode, e.g. "slow down") as well as low-level motion planning (the trajectory to follow). We assume an upstream perception module computes the scene context c – an object-oriented representation of the scene around the AV (also referred to as the *ego vehicle* or simply the *ego*) which includes kinematic information about the ego and the other agents, in addition to map and route information. The output of the planning module is a trajectory $\hat{\tau}$ that is passed to the downstream control module for tracking and execution.

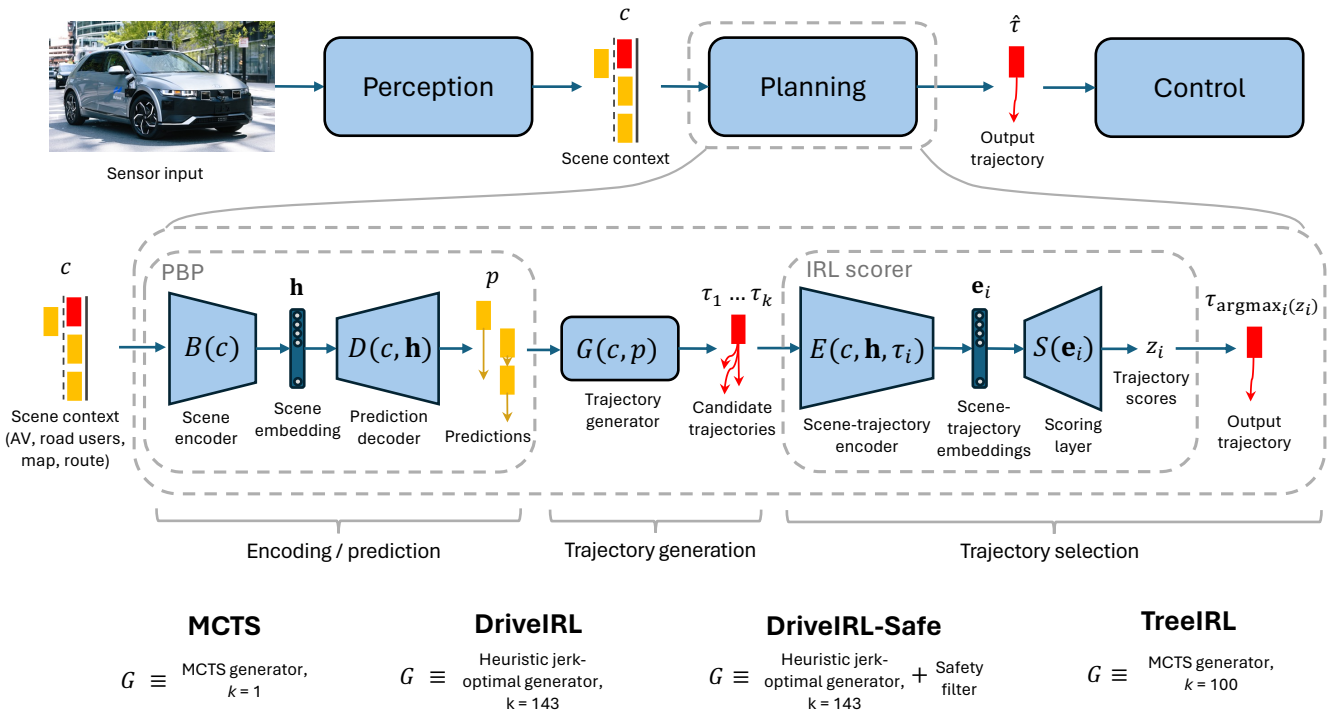


Fig. 3. Planner architecture (residual connections omitted).

Specifically, we use a version of the DriveIRL architecture (Fig. 3, middle) [24], 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 [32].
- 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.

TreeIRL uses the same planner architecture and components as DriveIRL, except for the generator G, which is replaced with MCTS (Fig. 2).

Inference. During inference, the trajectory with highest score is selected on each iteration:

$$\{\tau_1 \dots \tau_k\} \sim G(c, D(c, B(c)))$$
$$\hat{\tau} = \underset{\tau \in \{\tau_1 \dots \tau_k\}}{\arg\max}\ S(E(c, B(c), \tau))$$

Training. We pre-train PBP (B and D) for multi-agent motion prediction, as described previously [32]. We then keep those weights frozen and train the IRL scorer (E and S; G is not differentiable) on 80 hours of human expert driving using maximum-entropy IRL [24], [52], [53], formulated as a classification problem and optimized using a focal loss [54]. In particular, the probability of selecting the optimal trajectory is $P(\tau^*) = \frac{\exp(z^*)}{\sum_i \exp(z_i)}$, where τ^* is the optimal trajectory and z^* is its score (logit). Combining a negative log-likelihood loss with a focal loss gives the following loss for a single training sample:

$$loss = -(1 - P(\tau^*))^\gamma \log P(\tau^*),$$

where γ refers to the focusing parameter, not to be confused with the MDP discount factor (also denoted γ) elsewhere. For DriveIRL, τ^* corresponds to the trajectory from $\{\tau_1 \dots \tau_k\}$ that is closest to the human expert ground truth trajectory in L2 distance (position and velocity), excluding trajectories in collision with the future trajectories of other agents.

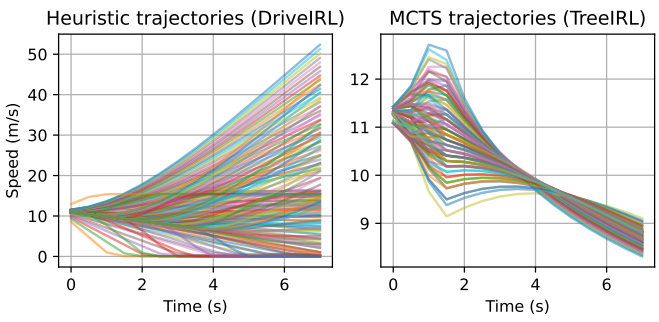


Fig. 4. Example open-loop trajectories on the same scenario.

TreeIRL is trained in the same way as DriveIRL, except that the L2 error decays exponentially for waypoints farther into the future. We put greater emphasis on the initial waypoints since 1) they are more critical for closed-loop behavior, 2) they correspond to actions that have been more thoroughly explored and evaluated by MCTS, 3) the predictions for those waypoints are likely to be more accurate, and 4) ground truth human expert behavior (during training) is more predictable in the short term. Overall, this allows us to circumvent a number of issues that arise when training with longer trajectories, such as mode collapse on scenarios in which the expert is reacting to future events that cannot possibly be anticipated at the present (e.g., the traffic light turning green, or the lead vehicle starting to move).

We also up-weigh the velocity L2 by 5x to place greater emphasis on matching the expert speed profile, which improves following behavior and comfort in the 1-D longitudinal domain considered here.

Limitations of enumerative trajectory generation. One of the key insights behind DriveIRL is that the planning problem can be simplified when separated into trajectory generation and selection in a fashion reminiscent of generative adversarial networks [55]. In particular, the trajectory generator does not have to be particularly sophisticated, as long as it provides sufficient coverage (i.e., maximum recall). In turn, the selection mechanism – the learned scoring function – merely has to discriminate between "good" and "bad" trajectories (i.e., maximum precision), rather than having to come up with the best trajectory from scratch.

However, in practice it is challenging to design a trajectory generator that provides sufficient coverage across a diverse set of scenarios. In particular, most trajectories output by the one-size-fits-all heuristic trajectory generator of DriveIRL are inappropriate for most scenarios, either accelerating or decelerating too much (Fig. 4, left). Even worse, sometimes all of the proposed trajectories are inappropriate and may even be ruled out by the safety filter, making it impossible for the scorer to find the needle in the haystack.

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. 4, right). For the rest of this section, we describe the technical details of the MCTS trajectory generator (Fig. 2).

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, particularly in real urban driving scenarios.

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:

$$a = \dddot{x}_{\text{ego}} \in \{-2, -1, 0, 1, 2\} \text{ m/s}^3$$

Transition function. The ego, lead agent, and static portions of the next state $s' \sim T(\cdot \mid s, a)$ are computed separately. The static components are updated as:

$$\begin{aligned} t' &= t + \Delta t \\ x'_{\text{max}} &= x_{\text{max}} \\ \dot{x}'_{\text{max}} &= \dot{x}_{\text{max}} \end{aligned}$$

For the ego, we use a simple kinematic model that forward integrates the jerk command to produce the next ego state:

$$\begin{aligned} \ddot{x}'_{\text{ego}} &= \text{clip}\big(\ddot{x}_{\text{ego}} + \dddot{x}_{\text{ego}}\Delta t,\ [-7 \text{ m/s}^2, 2 \text{ m/s}^2]\big), \\ \dddot{x}'_{\text{ego}} &= \frac{\ddot{x}'_{\text{ego}} - \ddot{x}_{\text{ego}}}{\Delta t}, \\ \dot{x}'_{\text{ego}} &= \max\Big(0,\ \dot{x}_{\text{ego}} + \ddot{x}_{\text{ego}}\Delta t + \tfrac{1}{2}\dddot{x}'_{\text{ego}}\Delta t^2\Big), \\ x'_{\text{ego}} &= \max\Big(x_{\text{ego}},\ x_{\text{ego}} + \dot{x}_{\text{ego}}\Delta t + \tfrac{1}{2}\ddot{x}_{\text{ego}}\Delta t^2 + \tfrac{1}{6}\dddot{x}'_{\text{ego}}\Delta t^3\Big), \end{aligned}$$

where $\dddot{x}'_{\text{ego}}$ is the effective jerk (not included in the state) that takes into account the acceleration clamp. We use a time step of $\Delta t = 0.5$ s. Note that driving in reverse is not allowed.

For the other agents, we use a non-reactive world model based on the (top mode) predictions p from PBP:

$$x'_{\text{lead}}, \dot{x}'_{\text{lead}}, \ddot{x}'_{\text{lead}} = \texttt{PREDICTED\_LEAD\_AGENT}(x'_{\text{ego}}, p, t')$$

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 world model, it significantly reduces latency compared to a reactive world model.

All transitions are deterministic, so for convenience we can write $s' \leftarrow T(s, a)$.

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

$$
\begin{aligned}
&R(s, a, s') = -\alpha\, cost(s')\text{, where} \\
&cost(s) \\
&= w_{\text{jerk}}\ \dddot{x}_{\text{ego}}^2 && (2)\\
&+ w_{\text{accel}}\ \ddot{x}_{\text{ego}}^2 && (3)\\
&+ w_{\text{speed}}\ |\dot{x}_{\max} - \dot{x}_{\text{ego}}| && (4)\\
&- 2\, w_{\text{speed}}\ \mathbb{I}\big[|\dot{x}_{\max} - \dot{x}_{\text{ego}}| < 0.5 \text{ m/s}\big] && (5)\\
&+ w_{\text{collision}}\ \mathbb{I}\big[x_{\text{ego}} \geq x_{\text{lead}}\big]\ (\dot{x}_{\text{lead}} - \dot{x}_{\text{ego}})^2 && (6)\\
&+ w_{\text{collision}}\ \mathbb{I}\big[x_{\text{ego}} \geq x_{\max}\big]\ \dot{x}_{\text{ego}}^2 && (7)\\
&+ w_{\text{clearance}}\ \mathbb{I}\big[0 < x_{\text{lead}} - x_{\text{ego}} < \delta\big]\ (x_{\text{lead}} - x_{\text{ego}} - \delta)^2 && (8)\\
&+ w_{\text{clearance}}\ \mathbb{I}\big[0 < x_{\max} - x_{\text{ego}} < \delta\big]\ (x_{\max} - x_{\text{ego}})^2 && (9)\\
&+ w_{\text{stop}}\ \mathbb{I}\big[\,\dot{x}_{\text{ego}} \approx 0 \wedge (\delta \leq x_{\text{lead}} - x_{\text{ego}} < 3\,\text{m})\big]\ (\dot{x}_{\max} - 2\dot{x}_{\text{ego}}) && (10)\\
&+ w_{\text{stop}}\ \mathbb{I}\big[\,\dot{x}_{\text{ego}} \approx 0 \wedge (0 \leq x_{\max} - x_{\text{ego}} < 2\,\text{m})\big]\ (\dot{x}_{\max} - 2\dot{x}_{\text{ego}}), && (11)
\end{aligned}
$$

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. 2 and 3 encourage comfort. Eq. 4 and 5 encourage (roughly) following the speed limit. Eq. 6 and 7 penalize collisions. Eq. 8 and 9 encourage maintaining a certain clearance. Finally, Eq. 10 and 11 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:

$$F(s) = \begin{cases} 1, & \text{if } t \geq H, \\ 0, & \text{otherwise.} \end{cases}$$

We use $H = 8$ s, which means that tree search and the rollouts never exceed a depth of 16.

B. MCTS components

We use a variant of MCTS based on the AlphaGo algorithm [46], [49] that can incorporate ML to guide the tree search towards more promising actions and thus drastically reduce 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 $\hat{V}_\theta$:

$$\big(\pi_\theta(a \mid s), \hat{V}_\theta(s)\big) = f_\theta(s, a),$$

where the parameters θ are learned using RL and/or IL. In our case, we can use f_θ in the MCTS trajectory generator in three distinct ways (Fig. 2):

- as a prior P in the tree policy to guide selection,
- as a rollout policy π_{rollout} and/or value function approximator $\hat{V}$ for leaf evaluation, and
- as a padding policy π_{padding} to generate trajectories from the top k leaves in the end.

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
    $v \leftarrow$ 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

Selection. For the tree policy, we use the PUCTS formula [46], which is an extension of the standard upper-confidence bound (UCB) algorithm [56] that uses the tree statistics Q and N to balance exploration – selecting unfamiliar nodes with low $N(s, a)$ – and exploitation – selecting rewarding nodes 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 placing greater weight on actions that are *a priori* promising – that is, before simulating their outcomes. For example, at the start of the tree search, when $Q(s, a) = 0, N(s, a) = 0\ \forall s \in \mathcal{S}, \forall a \in \mathcal{A}$, P will dictate which action is selected first.

In our experiments, we consider two possibilities for the prior policy P:

- the learned RL policy π_θ, i.e. $P(s, a) = \pi_\theta(a \mid s)$, or
- 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}:

$$\texttt{EVALUATE}(s) = \sum_{j=0}^{\infty} \gamma^j \, R(s_j, a_j, s_{j+1}) \, \mathbb{I}[F(s_j) = 0], \\ \text{where } a_j \sim \pi_{\text{rollout}}(\cdot \mid s_j), \; s_{j=0} = s.$$

We consider tree alternatives for π_{rollout}:

- The learned RL policy, i.e. $\pi_{\text{rollout}} \equiv \pi_\theta$,
- an IDM policy, i.e. $\pi_{\text{rollout}} \equiv \pi_{\text{IDM}}$, which deterministically chooses the acceleration of an IDM, and
- 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 a modified transition function that integrates 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. We experiment with three possibilities for the padding policy π_{padding} to generate trajectories from the top k leaves of the resulting tree:

- The learned RL policy, i.e. $\pi_{\text{padding}} \equiv \pi_\theta$,
- the IDM policy, i.e. $\pi_{\text{padding}} \equiv \pi_{\text{IDM}}$, and
- 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 $t = 0$ 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/yellow traffic light at which the ego can safely stop (whichever is closest) and remains the same during the tree search. The speed limit $\dot{x}_{\text{max}}$ comes from the map.

Since our focus is on longitudinal control, we assume that the ego is always on the reference path – i.e., that the lateral deviation is 0 m – and delegate any lateral correction to downstream post-processing.

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 ensuring kinematic feasibility.

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 using Proximal Policy Optimization (PPO) [57] within Stable-baselines3 [58] using a custom vehicle-following environment. The MDP of the RL agent is the same as the MDP used in MCTS. PPO hyperparameters include rollout lengths of 204,800 steps, batch size of 640, learning rate of 5×10^{-4}. During training, the RL agent observes a range of traffic scenarios, including lead vehicles maintaining constant speed, high-deceleration stop events, stop-and-go patterns, sudden cut-in maneuvers, and cases without a lead vehicle. The policy is trained for a total of 40 million steps. We find empirically that this configuration enables the RL agent to acquire stable and goal-directed behaviors across diverse driving scenarios, effectively balancing safety, smoothness, and progress toward the goal.

V. NuPlan experiments

We evaluate TreeIRL in nuPlan [4] against classical and state-of-the-art planners on scenarios based on real-world autonomous driving logs. 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.

NuPlan simulation. We use the open-source nuPlan simulator [4] to perform 10-Hz closed-loop simulations. The ego vehicle is propagated using a two-stage controller consisting of a Linear Quadratic Regulator (LQR) tracker [59], [60] followed by a kinematic bicycle model. The other agents are replayed from the log (log-playback). This approach has been favored by other authors [44], [61]–[63] since, by definition, it produces human-like behaviors for the other agents. However, log-playback is non-reactive, which can make the simulation result unrealistic, particularly if the ego vehicle deviates too much from the log (e.g., improbable rear collisions if it is slightly slower). We mitigate this by 1) using relatively short 20-s simulations (with 4 s of history), and 2) interpreting the metrics with caution – e.g., rear collisions are more indicative of progress and human-likeness rather than safety.

NuPlan dataset. We evaluate the planners on 7000+ scenarios corresponding to ∼40 hours of driving data collected in the Las Vegas metropolitan area by expert human drivers manually operating the AV. The dataset covers diverse geolocations with dense urban traffic at different times of day, including the Las Vegas Strip (68.29%), downtown (17.01%), airport (7.10%), and west of Strip (4.14%) areas. Scenarios span different behaviors of the ego and the other agents, such as starting from stationary (12%), decelerating (12%), cut-ins (3%), challenging cut-ins (2%), challenging ACC (3%), lead vehicle breaking (2%), remaining stationary (18%), as well as nominal driving (48%).

NuPlan metrics. We compute the following metrics:

- **Collisions**: instances when the ego bounding box first intersects that of another agent. We only count *at-fault* collisions, that is, collisions that could have been avoided if the ego had slowed down (roughly corresponding to front collisions).
- **Drivable area violations**: instances when the the ego bounding box is more than 0.3 m outside the mapped drivable area.
- **Traffic light violations**: instances when the ego crosses a stop line during a red light.
- **Speed limit violations**: continuous measure of how often the ego exceeds the speed limit (0 = no violations, 1 = significant violations).
- **Time gap**: minimum time gap (i.e., projected time-to-collision with nearby agents). Small values indicate close calls.
- **Progress along expert route**: ego progress along the route relative to the expert ground truth.
- **Comfort**: indicates whether the ego acceleration, yaw rate, and jerk remain within bounds empirically derived from the expert data: longitudinal accel $\in [-4.05, 2.40]$ m/s^2, absolute lateral accel ≤ 4.89 m/s^2, absolute yaw accel ≤ 1.93 rad/s^2, absolute yaw rate ≤ 0.95 rad/s, absolute longitudinal jerk ≤ 4.13 m/s^3, absolute jerk magnitude ≤ 8.37 m/s^3.
- **Min/max longitudinal jerk:** minimum and maximum longitudinal jerk (close to 0 is most comfortable).
- **Min/Max longitudinal accel**: minimum and maximum longitudinal accelerations (close to 0 is most comfortable).
- **L2 error:** average pointwise L2 distance between ego and expert trajectories.
- **Deceleration/acceleration delay error**: how much later the ego begins to decelerate/accelerate compared to the expert.
- **Max speed error**: maximum speed difference between the ego and the expert, normalized w.r.t. the maximum expert speed.

Metrics are averaged across simulations for each planner.

A. Tuning MCTS

We first compare different configurations of MCTS by taking only the top 1 trajectory (i.e., $k = 1$; Fig 2, bottom left), while treating the IRL scorer as pass-through. By effectively disabling the IRL scorer, we can evaluate MCTS on its own as a standalone planner, as is typically done. This allows us to explore the parameter space of the MCTS trajectory generator without having to retrain the IRL scorer for every MCTS configuration.

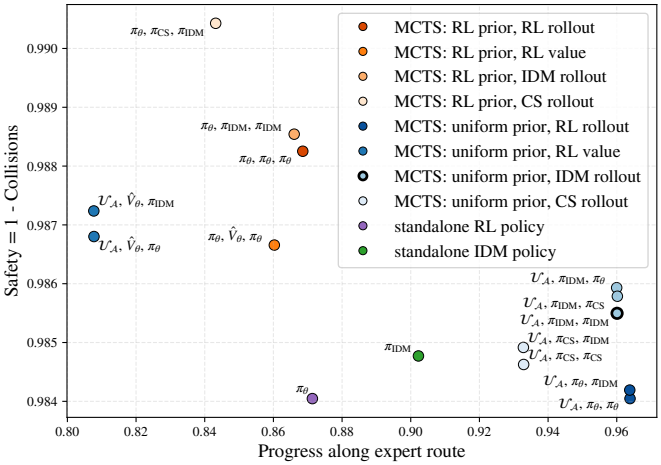


Fig. 5. Comparing MCTS configurations. Highlighted configuration chosen for subsequent experiments.

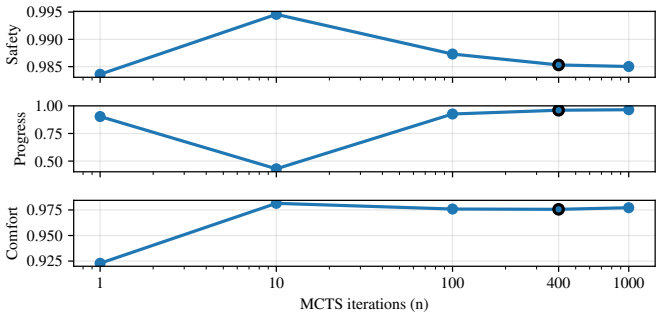


Fig. 6. Number of MCTS iterations. Highlighted setting chosen for subsequent experiments.

MCTS configurations. We explore different combinations of:

- prior policy P: RL policy (π_θ) or uniform policy ($\mathcal{U}_\mathcal{A}$),
- evaluation function: RL critic ($\hat{V}_\theta$) or RL rollout (π_θ) or IDM rollout (π_{IDM}) or constant speed rollout (π_{CS}),
- padding policy: RL (π_θ) or IDM (π_{IDM}) or constant speed (π_{CS}).

We also evaluate against the standalone RL (π_θ) and IDM (π_{IDM}) policies, which correspond to the special case of the MCTS generator when each of them is set the padding policy π_{padding} and the number of iterations is set to $n = 0$.

Progress vs. safety. With $n = 400$ iterations, using the RL policy either as a prior ($P \equiv \pi_\theta$) or as a rollout policy ($\pi_{\text{rollout}} \equiv \pi_\theta$) tends to produce more conservative behavior, with fewer collisions but also less progress (Fig. 5). Results are similar when using the RL critic $\hat{V}_\theta$. In contrast, a uniform prior ($P \equiv \mathcal{U}_\mathcal{A}$) results in more aggressive behavior overall, with more progress but also more collisions. With a uniform prior, the IDM rollout policy ($\pi_{\text{rollout}} \equiv \pi_{\text{IDM}}$) is strictly better than the constant speed rollout policy ($\pi_{\text{rollout}} \equiv \pi_{\text{CS}}$).

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

* – configuration chosen for subsequent experiments.

Iterations. With a uniform prior ($P \equiv \mathcal{U}_\mathcal{A}$), IDM rollout ($\pi_{\text{rollout}} \equiv \pi_{\text{IDM}}$), and IDM padding ($\pi_{\text{padding}} \equiv \pi_{\text{IDM}}$), performance plateaus around $n = 400$ iterations (Fig. 6).

Latency. Real-world deployment makes latency a critical consideration, as the benefit of an improved planner can be lost if it is too slow to react. We compare the latency of different MCTS trajectory generators with $k = 100$ trajectories – as they would be used with IRL – on 100 scenarios each. Latency is measured on a Lenovo ThinkPad laptop running Ubuntu 22.04 with an Intel Core i9-10885H CPU (2.40 GHz base, up to 5.30 GHz turbo, 8 cores/16 threads) and 16 MB L3 cache, restricted to a single CPU thread (as on the car).

Relying on the learned policy π_θ for the rollouts is around two orders of magnitude slower than relying on the IDM policy π_{IDM} (Tab. I). For padding, the difference is around one order of magnitude. Our goal is to run the planner on the car at 10 Hz or more, so any latency above 100 ms is unacceptable. This effectively excludes configurations with $\pi_{\text{rollout}} \equiv \pi_\theta$ or $\pi_{\text{padding}} \equiv \pi_\theta$.

Intermediate discussion. The relatively conservative behavior of the learned policy π_θ is likely due to the low-dimensional state space, which only captures the ego and lead vehicles. Since the RL policy is model-free, this is the only information it can use to make decisions. As a result, the RL policy is effectively "blind" to any agent that is not immediately in front of it, until that agent actually appears. Because of this, the RL policy cannot anticipate, for example, that an agent would cut in from an adjacent lane; so in order to avoid collisions, it converges on more conservative behavior overall.

In contrast, MCTS is model-based and, in our case, it can use the predictions to evaluate the consequences of its actions. As a result, despite using the same low-dimensional state space, MCTS can anticipate cut-ins and other kinds of interactions. For example, if a cut-in is unlikely, it will evaluate actions that make progress as more rewarding; conversely, if a cut-in is likely, it will prefer conservative actions that avoid collisions. This leads to improved progress and safety compared to the standalone RL and IDM planners.

Given the latency costs of the learned policy π_θ and its overly conservative bias (as well as that of $\hat{V}_\theta$), we use the following parameters for the MCTS generator in subsequent experiments: $n = 400, k = 100, P \equiv \mathcal{U}_\mathcal{A}, \pi_{\text{rollout}} \equiv \pi_{\text{IDM}}, \pi_{\text{padding}} \equiv \pi_{\text{IDM}}$.

B. TreeIRL evaluation

Baselines. We evaluate TreeIRL against the following following classical and state-of-the-art baselines (Fig. 1):

- The intelligent driver model (**IDM**) [26]: 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 (Fig. 3, bottom left),
- Path-based prediction (**PBP**) [32]: an open-loop motion forecasting model trained using imitation learning (Fig. 3, middle left),
- **DriveIRL** [24]: 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 (Fig. 3, bottom middle),
- **Gigaflow** [40]: a RL-based motion planner trained in closed loop using self-play, with the same policy controlling the ego and all other agents (we use an in-house implementation based on [40], as there is no code/weights available),
- **DriveIRL-Safe** [24]: identical to DriveIRL, with the addition of a safety filter as described in [24], [43] that excludes any apparently unsafe trajectories (Fig. 3, bottom middle).

Results. Most planners exhibit comparable and acceptable safety and progress (Tab. II, Fig. 7), 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.

Since the collision metric appears saturated across all scenarios, we further zoom in on a subset of challenging cut-in cases where an agent appears abruptly at a short distance in front of the ego (Tab. III). On this set, DriveIRL-Safe and TreeIRL have around half as many collisions as MCTS and DriveIRL, while maintaining the same level of progress. While safety and similarity to the human expert driver is better for DriveIRL-Safe compared to TreeIRL, comfort is substantially worse.

VI. 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

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) $\downarrow\vert$	0.32	0.31	0.42	**0.14**	0.78	0.15	0.43
	Acceleration delay error (s) $\vert\downarrow$	**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

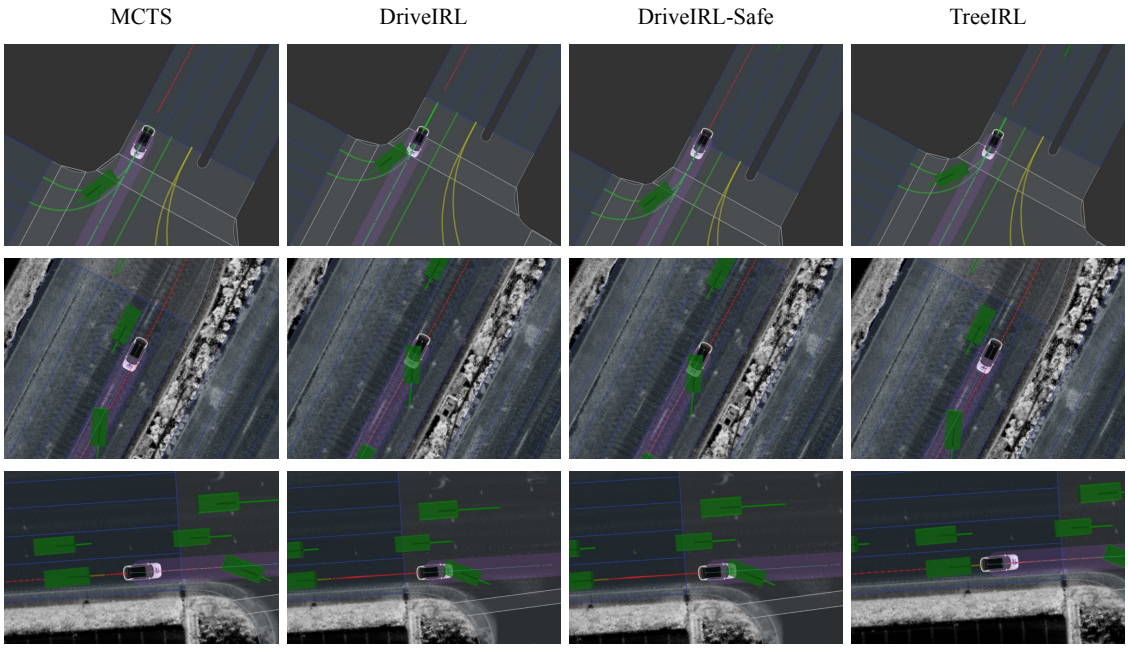


Fig. 7. NuPlan examples of DriveIRL collisions avoided by TreeIRL. Each row corresponds to a different scenario. Columns correspond to snapshots of the same moment in time for different planners.

TABLE III
NUPLAN EVALUATION ON CHALLENGING CUT-INS (134 SCENARIOS)

Metric	MCTS	DriveIRL	DriveIRL-Safe	TreeIRL
Collisions ↓	0.31	0.34	**0.16**	0.19
Drivable area violations ↓	**0.01**	**0.01**	**0.01**	**0.01**
Traffic light violations ↓	**0.00**	0.01	0.01	**0.00**
Speed limit violations ↓	0.17	0.16	0.14	**0.13**
Time gap (s) ↑	0.72	0.67	**1.02**	1.01
Progress along expert route ↑	**0.99**	0.98	0.98	0.98
Comfort ↑	0.84	0.86	0.75	**0.87**
L2 error (m) ↓	6.60	5.50	**3.30**	5.28

planners on Hyundar IONIQ 5 self-driving cars and evaluate them on public roads in Las Vegas.

High-fidelity simulation. As in previous work [19], prior to on-road deployment, performance of the planning system is thoroughly evaluated using the Object Sim simulator from Applied Intuition [64], which performs realistic high-fidelity physics simulation of vehicle dynamics. Unlike nuPlan, which is Python-based, only simulates the planner, and uses a simplified simulator, Object Sim simulates the entire AV stack in a more realistic setting closely resembling real-world deployment. This allows us to get a better sense of on-road performance and to catch integration issues that may not appear in nuPlan. As in nuPlan, other agents are replayed from the log. All simulations are in closed loop, last 30 s (including 4 s of warmup), and run at 10 Hz.

Overall, nuPlan can be viewed as a cheap and fast way to get an initial sense of planner performance, making it useful for parameter tuning and for ruling out obviously inferior planners. In contrast, Object Sim is a robust yet computationally intensive way to get a more realistic sense

of planner performance in the context of a full AV stack.

Simulation scenarios. We use a set of 600+ handpicked scenarios based on on-road events in Las Vegas metropolitan area that were particularly challenging for the AV, including close ACC (5%), encroachments (2%), cut-ins (2%), traffic lights (7%), low speed (6%), high speed (11%), far gap behind lead (15%), harsh braking (17%), lane biasing (6%), jerky driving (24%).

On-road scenarios. Routes cover similar geolocations to the simulations, including the Strip, downtown, and west of Strip areas. Driving in Las Vegas involves navigating dense urban traffic, handling traffic lights and intersections, smoothly following and breaking for other drivers, responding safely and comfortably to cut-ins and lead vehicle breaking. All on-road tests are performed with an experienced safety driver ready to take over immediately in case of unsafe driving, as well as an experienced test engineer continuously monitoring AV performance.

Simulation metrics. We use a similar set of metrics to nuPlan, with some differences largely due to the differences in the simulators and the data.

- **Front collisions**: instances when the ego collides at the front while moving (speed > 0.01 m/s).
- **Traffic light violations**: instances when the ego fully crosses the stop line of a red traffic light.
- **Stop line violations**: instances when the ego partially crosses the stop line of a red traffic light.
- **Speed limit violation time**: total time spent driving above the speed limit.
- **ACC distance violations**: instances when the following distance to the lead vehicle drops below 1.5 m.
- **Min time gap**: minimum time gap to the lead vehicle. Small numbers indicate close calls.
- **Average speed**: average ego speed.
- **Rear collisions**: instances when the ego is hit from behind. Due to the non-reactive simulation, these are interpreted as a measure of progress: rear collisions indicate slower driving / not keeping with the flow of traffic.
- **Brake taps**: instances when the negative longitudinal jerk (brake) exceeds 4.25 m/s^3.
- **Longitudinal jerk violations**: instances when the absolute longitudinal jerk exceeds 5 m/s^3.
- **Lateral jerk violations**: instances when the absolute lateral jerk exceeds 7 m/s^3.

As in nuPlan, simulation metrics are reported as averages across simulations.

On-road metrics. Metrics that are useful in simulation may not be practical when assessing real-world driving. For example, collisions will always be 0, as the safety driver will take over before any real collision actually occurs. We therefore measure on-road performance with a combination of discretionary and objective metrics. Discretionary metrics correspond to events manually tagged at the discretion of the test team onboard the AV during road tests. These include takeovers by the safety driver when AV behavior is deemed unsafe, as well as manually flagged events corresponding to comfort or progress issues. Objective metrics are computed after-the-fact based on data logged by the AV.

TABLE IV
DEPLOYMENT STRATEGY LATENCY MEASUREMENTS

Statistic	DriveIRL + Lab2Car (ms)	DriveIRL + Smoother (ms)
Max	246.70	26.16
P99.99	246.70	26.16
P99	208.81	20.95
P50	60.23	14.29
Average	73.09	14.63

We use the following discretionary metrics:

- **Safety takeovers**: any takeover by the safety driver due to unsafe AV behavior.
- **ACC failures**: safety takeover preventing collisions with a lead vehicle.
- **Traffic light failures**: safety takeovers preventing traffic light violations.
- **Cut-in failures**: safety takeovers preventing collisions due to cut-ins.
- **Slow driving**: instances when the AV drives too slowly.
- **Discomfort**: any instance of uncomfortable driving (jerky or discomfort braking).
- **Jerky driving**: instance of jerky driving, e.g. brake taps or jerky throttle.
- **Harsh/mild discomfort braking**: instances of severe/light uncomfortable braking.

We use the following objective metrics:

- **Harsh/mild discomfort braking**: output of classifier trained to predict human-annotated harsh/mild discomfort braking based on vehicle kinematics.
- **Brake taps**: identical to brake taps in simulation.

All on-road metrics are reported as autonomous driving miles (auto-miles/auto-mi) per event.

A. Deployment strategy

Like most ML-based planners, DriveIRL and DriveIRL-Safe do not ensure kinematic feasibility. MCTS and TreeIRL ensure kinematic feasibility only longitudinally. Neither planner ensures dynamic feasibility. This necessitates performing some kind of post-processing on the output trajectory $\hat{\tau}$ before passing it to the downstream drive-by-wire system for execution.

We consider two deployment strategies:

- Lab2Car [19]: a two-step process that converts $\hat{\tau}$ into a set of spatiotemporal constraints – a "maneuver" – that incorporate information about vehicle dynamics, road geometry, and (optionally) other agents (although we do not take advantage of that feature). The resulting optimization problem is solved using an industry-grade MPC solver.
- Smoother: a lightweight MPC that applies minor kinematic corrections to $\hat{\tau}$ and performs bookkeeping to ensure continuity between planning cycles. The optimization is performed using a kinematic bicycle model in Cartesian 2D coordinates.

TABLE V
COMPARING DEPLOYMENT STRATEGIES

Category	Metric	DriveIRL		MCTS	
		+ Lab2Car	+ Smoother	+ Lab2Car	+ Smoother
High-fidelity 30-s simulations					
	Total simulations	605	605	605	605
Safety	Front collisions ↓	**0.05**	**0.05**	**0.05**	**0.05**
	Speed limit violation time (s) ↓	**4.14**	5.76	3.94	**3.68**
	ACC distance violations (< 1.5 m) ↓	**0.20**	0.25	**0.19**	**0.19**
	Min time gap (s) ↑	**1.26**	1.01	**1.48**	1.43
Progress	Average speed (m/s) ↑	**8.32**	8.23	**9.82**	9.79
Comfort	Brake taps ↓	**0.18**	0.29	0.29	**0.27**
	Longitudinal jerk violations ↓	**0.12**	0.17	0.29	**0.23**
	Longitudinal accel violations ↓	**0.23**	0.29	0.54	**0.50**
Public road evaluation					
	Total auto-miles	40.8	34.8	62.1	64.1
Comfort	Harsh discomfort braking (mi/event) ↑	**40.8**	8.70	5.17	**9.16**
	Mild discomfort braking (mi/event) ↑	**40.8**	11.60	5.17	**8.01**
	Brake taps (mi/event) ↑	**1.57**	0.87	0.60	**1.49**

Deployment strategies compared separately for DriveIRL and MCTS.

We first deploy DriveIRL with each strategy and measure the system latency using open-loop resimulations on identical logs using identical hardware as on the car (Tab. IV). Using the Smoother is about an order of magnitude faster than using Lab2Car. However, subsequent closed-loop simulations and real-world driving (Tab. V) show that DriveIRL + Lab2Car performs better across all metrics compared to DriveIRL + Smoother. Interestingly, we observe the opposite pattern for MCTS.

Interim discussion. We suspect MCTS fails to benefit from Lab2Car due to its bookkeeping mechanism and jerk controls, which make it sensitive to latency, particularly in an asynchronous environment. For example, consider the case when MCTS commands braking with jerk -2 m/s^3 from cruising speed. Initially, the deceleration will be small; this will be sent to Lab2Car, which will command a gentle brake. However, while Lab2Car is still running, MCTS (which is must faster and hence runs at higher frequency) will run for several iterations, each commanding jerk -2 m/s^3 and assuming that the previous jerk command was executed (due to bookkeeping). MCTS will thus quickly reach the max decel of -7 m/s^2. At the next cycle when Lab2Car finally catches up, the MCTS trajectory will now appear to suddenly command a harsh brake, leading to discomfort.

In contrast, DriveIRL plans from the measured pose, making it more robust to downstream delays. However, this means it may also be less reactive.

For all subsequent experiments, we deploy DriveIRL/DriveIRL-Safe with Lab2Car and MCTS/TreeIRL with the Smoother.

B. TreeIRL evaluation

We integrate MCTS, DriveIRL, DriveIRL-Safe, and TreeIRL with the Motional AV-stack (Fig. 3) and compare them in simulation, using our full-stack simulator, and in the real world, using Hyundar IONIQ 5 self-driving cars (Tab. VI, Fig. 8 & 9).

High-fidelity 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. 8) show that DriveIRL fails to stop on time to prevent collisions, while DriveIRL-Safe stops too soon and too abruptly. Together, these results validate the conclusions from the nuPlan simulations and the choice of deployment strategy.

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. 9) 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.

Interim discussion. The substantially larger advantage of TreeIRL over the other planners 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 perceived safety, takeovers better reflect the subjective experience of the rider, which in some sense is the ultimate yardstick for measuring AV performance. The sudden braking during takeovers addition-

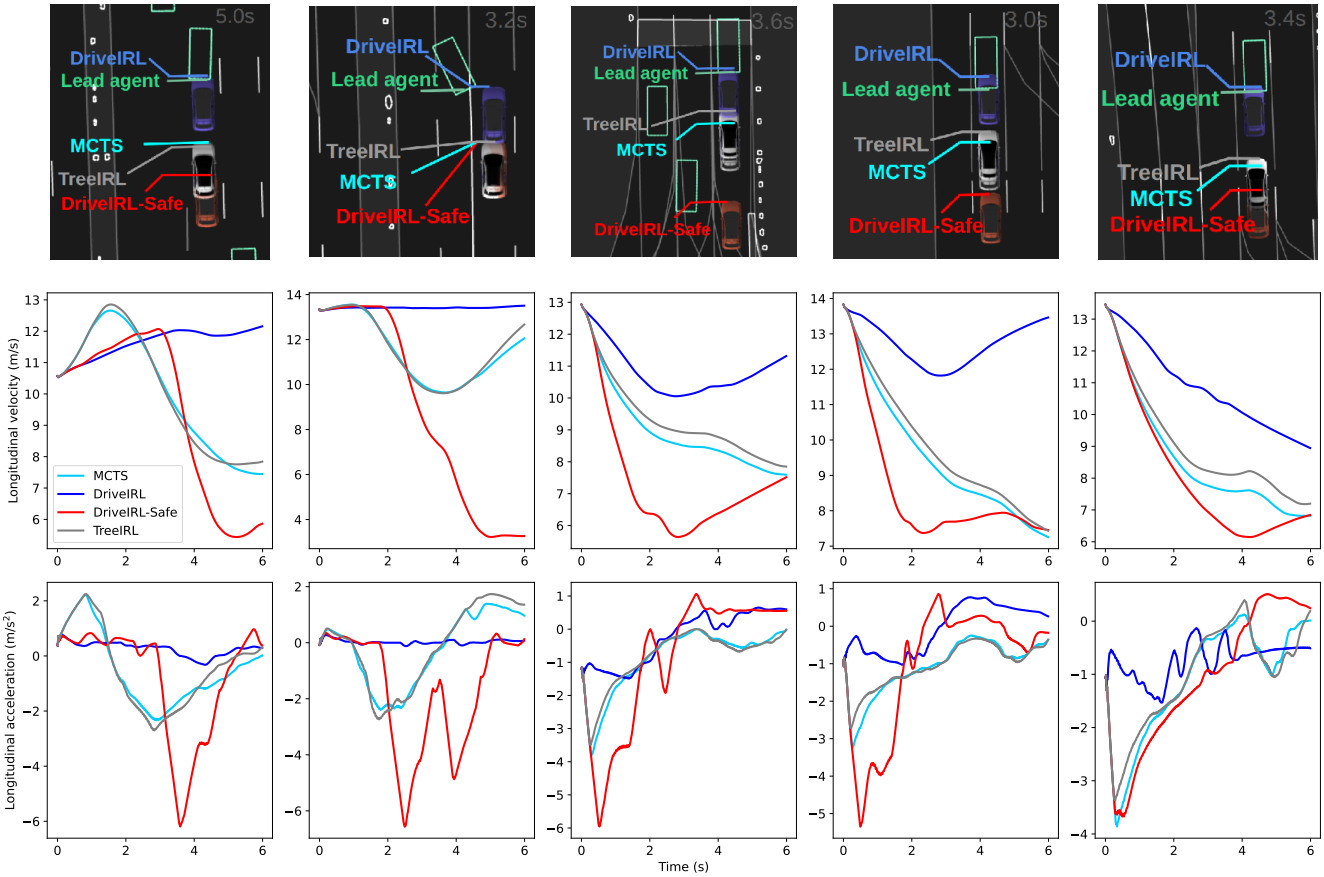


Fig. 8. High-fidelity simulation examples. Each column corresponds to a different scenario. Top row, snapshot of the same moment in time in the simulation for each planner.

Fig. 9. Cut-in examples from real-world driving. Right inset, recent history of AV kinematics. For DriveIRL, $t = 0$ corresponds to a safety-related takeover (cut-in failure).

TABLE VI
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**

ally impacts comfort, which is likely the main factor behind the worse comfort of DriveIRL on the road compared to the simulations.

It is worth noting that the on-road results for DriveIRL and DriveIRL-Safe are in line with previous reports [24] of 2 takeovers for 8.8 auto-miles and 0 takeovers for 6.9 auto-miles, respectively, albeit on a much smaller evaluation.

VII. GENERAL DISCUSSION AND CONCLUSIONS

We presented TreeIRL, a novel combination of MCTS and IRL for autonomous driving that outperforms the state of the art in simulated and real-world urban driving in terms of safety, progress, comfort, and human-likeness. To the best of our knowledge, this is the first real-world evaluation of a motion planner based on MCTS. We suspect this is due to latency: since the size of the search space (i.e., number of possible trajectories) is exponential in the size of the state space, most applications of MCTS to motion planning require more iterations to converge than is practically feasible on a real AV. One solution is to use a learned policy [49]; however, this requires performing inference-in-the-MCTS-loop – potentially multiple times during the rollouts – which further strains latency. While this can be overcome with more compute and parallelization [46], this is not always feasible on board a real AV where multiple systems compete for scarce resources in real time.

Instead, by combining MCTS with IRL, we can relax the demands on the MCTS generator, as it no longer needs to find the single best trajectory but merely to home in on a promising subset of the trajectory space and lean on the IRL scorer to choose the best trajectory. One way to view this is that MCTS handles safety, by choosing a safe behavior mode, while IRL handles comfort, by choosing the most human-like variation around that mode. Indeed, our simulation (Tab. II) and on-road (Tab. VI) results indicate that the biggest gain of TreeIRL over vanilla MCTS is in comfort, although safety is also improved.

On the road, TreeIRL showed close to 2 orders the magnitude better safety than DriveIRL (Tab. VI), highlighting the benefits of combining classical and learning-based approaches. Accordingly, the advantage of TreeIRL over DriveIRL-Safe – another hybrid system – was smaller. However, DriveIRL-Safe was inferior to both TreeIRL and DriveIRL in terms of comfort and progress. This highlights the limitations of naïve enumerative trajectory generation, which can fail to provide sufficient coverage of the trajectory space: once the safety filter rules out apparently "bad" trajectories, there may be insufficient diversity in the remaining trajectories to ensure comfort and progress.

The emergence of public simulators and benchmarks such as nuPlan [4], CARLA [65], and Waymax [66], [67] has been instrumental in advancing motion planning research by facilitating the comparison of different planners on an equal footing. However, compressing planner performance into a single scalar score can obfuscate critical planner limitations

(Tab. II). Until a single universal gold standard metric is designed and agreed upon, we believe planners should be compared holistically across a diverse set of metrics capturing safety, progress, and comfort (Fig. 1, bottom right).

Our work also highlights the limitations of relying solely on simulation to assess planner performance. The discrepancy between simulated and real-world performance – the sim-to-real gap – is well-documented [38], [68], yet it is nevertheless striking that the relatively small advantage of TreeIRL in simulation (Tab. II) translated to 1-2 orders of magnitude improvement on the road (Tab. VI). Until simulation fidelity improves to reliably match on-road performance – a problem nearly as hard as the planning problem itself – we believe early on-road tests [19] should be a critical part of comprehensive planner evaluation. Thus planner comparison should proceed both holistically and incrementally, with a broad set of experimental planners evaluated in simulation on a broad set of metrics and then narrowed down to a smaller set of promising planners that are deployed and evaluated in the real world.

An alternative way to combine MCTS with IRL would be to replace the handcrafted reward function R (Eq. 2-11) with a reward function learned from data. This learned reward function can be directly plugged into MCTS and/or used to train the RL network f_θ. This could obviate the need for a learned trajectory scorer, as the reward function itself would already capture human-likeness. Yet another way to rid of the IRL scorer would be to train f_θ with a hybrid of RL and IL [44].

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 [39] or Gigaflow [40]. Gigaflow could also facilitate the expansion of the state/action space to include lateral control by replacing the RL network f_θ, as planning in 2D instead of 1D would likely require prohibitively many iterations if MCTS is guided by a simple policy such as the IDM. This would enable new capabilities such as lane biasing and lane changes, which we leave as the subject of future work. 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; and the countless other Motional scientists and engineers who contributed to this project directly or indirectly.

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] S. Vora, A. H. Lang, B. Helou, and O. Beijbom, "Pointpainting: Sequential fusion for 3d object detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020, pp. 4604–4612.

[12] 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.

[13] 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.

[14] Q. Chen, S. Vora, and O. Beijbom, "Polarstream: Streaming object detection and segmentation with polar pillars," in *Proceedings of the 34th Conference on Neural Information Processing Systems (NeurIPS)*, 2021.

[15] M. Liang, B. Yang, R. Hu, Y. Chen, R. Liao, S. Feng, and R. Urtasun, "Learning lane graph representations for motion forecasting," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2020, pp. 541–556.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] S. J. Qin and T. A. Badgwell, "A survey of industrial model predictive control technology," *Control engineering practice*, vol. 11, no. 7, pp. 733–764, 2003.

[28] R. Rajamani, *Vehicle dynamics and control*. Springer, 2006.

[29] C. Liu, S. Lee, S. Varnhagen, and H. E. Tseng, "Path planning for autonomous vehicles using model predictive control," in *2017 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2017, pp. 174–179.

[30] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[31] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of monte carlo tree search methods," *IEEE Transactions on Computational Intelligence and AI in games*, vol. 4, no. 1, pp. 1–43, 2012.

[32] 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.

[33] S. Ross, G. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, 2011, pp. 627–635.

[34] P. De Haan, D. Jayaraman, and S. Levine, "Causal confusion in imitation learning," *Advances in neural information processing systems*, vol. 32, 2019.

[35] X. Pan, Y. You, Z. Wang, and C. Lu, "Virtual to real reinforcement learning for autonomous driving," *arXiv preprint arXiv:1704.03952*, 2017.

[36] X. Liang, T. Wang, L. Yang, and E. Xing, "Cirl: Controllable imitative reinforcement learning for vision-based self-driving," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 584–599.

[37] Z. Zhang, A. Liniger, D. Dai, F. Yu, and L. Van Gool, "End-to-end urban driving by imitating a reinforcement learning coach," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2021, pp. 15 222–15 232.

[38] 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.

[39] 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.

[40] 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.

[41] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," in *Proceedings of the twenty-first international conference on Machine learning*, 2004, p. 1.

[42] Z. Huang, J. Wu, and C. Lv, "Driving behavior modeling using naturalistic human driving data with inverse reinforcement learning," *IEEE transactions on intelligent transportation systems*, vol. 23, no. 8, pp. 10 239–10 251, 2021.

[43] M. Tomov, E. Wolff, and S. Omari, "Safety filter for machine learning planners," US Patent US20 230 373 529A1, 2023, uS Patent App. 18/320,899. [Online]. Available: https://patents.google.com/patent/US20230373529A1/en

[44] 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.

[45] M. Vitelli, Y. Chang, Y. Ye, A. Ferreira, M. Wołczyk, B. Osiński, M. Niendorf, H. Grimmett, Q. Huang, A. Jain *et al.*, "Safetynet: Safe planning for real-world self-driving vehicles using machine-learned policies," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2022.

[46] 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.

[47] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel *et al.*, "A general reinforcement learning algorithm that masters chess, shogi, and go through self-play," *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018.

[48] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel *et al.*, "Mastering atari, go, chess and shogi by planning with a learned model," *Nature*, vol. 588, no. 7839, pp. 604–609, 2020.

[49] 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.

[50] R. Chekroun, T. Gilles, M. Toromanoff, S. Hornauer, and F. Moutarde, "Mbappe: Mcts-built-around prediction for planning explicitly," in *2024 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2024, pp. 2062–2069.

[51] R. S. Sutton, A. G. Barto *et al.*, *Reinforcement learning: An introduction*. MIT press Cambridge, 2018, vol. 1, no. 1.

[52] 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.

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

[54] 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.

[55] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," *Advances in neural information processing systems*, vol. 27, 2014.

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

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

[58] 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

[59] J. Liu, Z. Yang, Z. Huang, W. Li, S. Dang, and H. Li, "Simulation performance evaluation of pure pursuit, stanley, lqr, mpc controller for autonomous vehicles," in *2021 IEEE international conference on real-time computing and robotics (RCAR)*. IEEE, 2021, pp. 1444–1449.

[60] B. Varma, N. Swamy, and S. Mukherjee, "Trajectory tracking of autonomous vehicles using different control techniques (pid vs lqr vs mpc)," in *2020 International conference on smart technologies in*

computing, electrical and electronics (ICSTCEE). IEEE, 2020, pp. 84–89.

[61] P. Kothari, C. Perone, L. Bergamini, A. Alahi, and P. Ondruska, "Drivergym: Democratising reinforcement learning for autonomous driving," *arXiv preprint arXiv:2111.06889*, 2021.

[62] E. Vinitsky, N. Lichtlé, X. Yang, B. Amos, and J. Foerster, "Nocturne: a scalable driving benchmark for bringing multi-agent learning one step closer to the real world," *Advances in Neural Information Processing Systems*, vol. 35, pp. 3962–3974, 2022.

[63] Q. Li, Z. Peng, L. Feng, Q. Zhang, Z. Xue, and B. Zhou, "Metadrive: Composing diverse driving scenarios for generalizable reinforcement learning," *IEEE transactions on pattern analysis and machine intelligence*, vol. 45, no. 3, pp. 3461–3475, 2022.

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

[65] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "Carla: An open urban driving simulator," in *Conference on robot learning*. PMLR, 2017, pp. 1–16.

[66] S. Ettinger, S. Cheng, B. Caine, C. Liu, H. Zhao, S. Pradhan, Y. Chai, B. Sapp, C. R. Qi, Y. Zhou *et al.*, "Large scale interactive motion forecasting for autonomous driving: The waymo open motion dataset," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2021, pp. 9710–9719.

[67] C. Gulino, J. Fu, W. Luo, G. Tucker, E. Bronstein, Y. Lu, J. Harb, X. Pan, Y. Wang, X. Chen *et al.*, "Waymax: An accelerated, data-driven simulator for large-scale autonomous driving research," *Advances in Neural Information Processing Systems*, vol. 36, pp. 7730–7742, 2023.

[68] 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.