2023 IEEE International Conference on Robotics and Automation (ICRA 2023)
May 29 - June 2, 2023. London, UK

DriveIRL: Drive in Real Life with Inverse Reinforcement Learning

Tung Phan-Minh, Forbes Howington, Ting-Sheng Chu, Momchil S. Tomov, Robert E. Beaudoin, Sang Uk Lee, Nanxiang Li, Caglayan Dicle, Samuel Findler, Francisco Suarez-Ruiz, Bo Yang, Sammy Omari*, Eric M. Wolff*
Motional AD Inc.
{tung.phan, forbes.howington, ting-sheng.chu}@motional.com

2023 IEEE International Conference on Robotics and Automation (ICRA) | 979-8-3503-2365-8/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICRA48891.2023.10160449

***Abstract*—In this paper, we introduce the first published planner to drive a car in dense, urban traffic using Inverse Reinforcement Learning (IRL). Our planner, DriveIRL, generates a diverse set of trajectory proposals and scores them with a learned model. The best trajectory is tracked by our self-driving vehicle's low-level controller. We train our trajectory scoring model on a 500+ hour real-world dataset of expert driving demonstrations in Las Vegas within the maximum entropy IRL framework. DriveIRL's benefits include: a simple design due to only learning the trajectory scoring function, a flexible and relatively interpretable feature engineering approach, and strong real-world performance. We validated DriveIRL on the Las Vegas Strip and demonstrated fully autonomous driving in heavy traffic, including scenarios involving cut-ins, abrupt braking by the lead vehicle, and hotel pickup/dropoff zones. Our dataset, a part of nuPlan, has been released to the public to help further research in this area.**

***Index Terms*—ML-based Planning, Inverse Reinforcement Learning, Real-World Deployment, Self-Driving, Autonomous Vehicles, Urban Driving, Learning from Human Driving.**

I. Introduction

Self-driving cars have been the focus of significant research and development over the past decade. Some companies are tantalizingly close to deploying a commercial self-driving taxi service that would make urban transportation cheaper and safer. Progress in self-driving cars has been largely driven by new datasets [1]–[4] that helped fuel dramatic improvements in ML approaches to object detection [5], [6] and motion forecasting [7]–[9]. However, the critical motion planning and decision-making algorithms that ultimately determine driving behavior have yet to see similar benefits from machine learning.

Classical planning and decision-making algorithms for self-driving cars rely heavily on hand-engineered components [10]. Developers will typically hand-tune the scoring function that determines which behaviors are desirable. Manually adjusting features and weights is a painstaking process with improvement in one area often causing unintended regressions elsewhere. Our planner obviates the need to craft detailed features or tune weights by learning these components from expert demonstrations using a maximum entropy IRL framework.

Our DriveIRL system uses simple and interpretable modules to do the relatively easy tasks of generating a diverse set of ego trajectories and optionally checking that they satisfy basic safety requirements. Careful construction of the proposed trajectories ensures that they a) are dynamically feasible, b) follow the route, c) satisfy assumptions from the vehicle controller, and d) are diverse. The learning component of our model focuses entirely on scoring these trajectories based on expert demonstrations. Our formulation directs the model capacity towards hard-to-specify nuances in behavior (e.g., speed profiles, clearances) more than creating "nice" trajectories and avoiding obviously unsafe behavior.

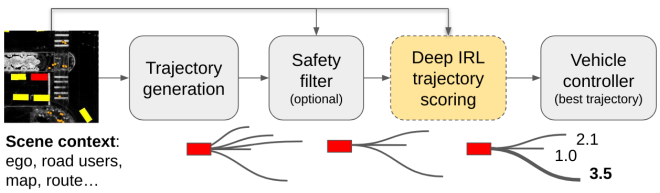


Fig. 1. DriveIRL architecture. The learned scoring component is indicated with a dotted boundary.

DriveIRL achieves strong real-world driving performance on the Las Vegas Strip, a major thoroughfare which connects many major hotels and casinos. Challenges include dense traffic, aggressive cut-ins, erratic drivers, and busy passenger pickup/dropoff zones near the hotels. We deployed DriveIRL on a car which drove fully autonomously on the Strip in these scenarios, showing the practical utility of our approach.

Our main contributions towards learning-based planning for self-driving cars are:

- The first, to our knowledge, IRL-based planner to drive a car in dense, urban traffic.
- A simple yet powerful architecture based on intuitive, minimally hand-engineered features that 1) learns the most-challenging-to-specify aspect of driving, 2) promotes creativity in input design at the trajectory level, and 3) lightens the engineering burden as the final feature scores are learned automatically, a sharp contrast to standard linear IRL-based methods that only learn feature (importance) weights and rely on handcrafting feature scores.
- Detailed evaluation of our planner on nuPlan, a real-world dataset that has been released to the public.

II. Related Work

Classical planning: Traditional approaches formulate the planning problem as search over an appropriately constructed graph (e.g., A*, RRT*, PRM*; [10], [11]) or trajectory optimization [10]. These methods often have strong theoretical

*Work done while at Motional.

guarantees on convergence to an optimal solution and are relatively easy to interpret. However, the cost function that defines desired behavior is often hand-engineered, and in practice requires painstakingly detailed tuning.

Imitation learning (IL): IL methods attempt to directly imitate the actions of an expert driver and have seen promising applications to self-driving cars pioneered by the work of ALVINN [12]. More recently, an end-to-end policy was learned from camera images for lane keeping [13].

A fundamental issue with IL is that there is a distribution shift from training to deployment with small errors causing the model to operate outside of its training data, which then leads to larger errors. ChauffeurNet [14] uses behavioral cloning with extensive data augmentation to mitigate the distribution shift issue, and UrbanDriver [15] uses an offline policy gradient method with closed-loop rollouts during training to automatically create appropriate data augmentation. While data augmentation improves our performance, it is not as critical since our trajectory generation mechanism can pull the car towards the lane center.

Another related work by [16] learns a non-interpretable temporal-spatio costmap over the environment to indirectly score a set of procedurally generated trajectories. Our approach, in contrast, learns to score trajectories directly, thereby avoiding the assumption of an additive spatial-temporal costmap altogether. We also improve on trajectory generation by guaranteeing map compliance and demonstrate the effectiveness of our model on a vehicle in dense urban traffic.

Another similar approach is that of [17], where a hybrid model with a learned planner and an interpretable fallback layer drives in San Francisco. Our IRL-based model is simpler and less reliant on a fallback layer. Furthermore, the recursive check of our safety filter is less conservative.

Reinforcement learning (RL): RL approaches learn a driving policy by optimizing a reward function. The standard approach requires a simulator (e.g., CARLA; [18]) to update the environment that the driving policy interacts with. There have been a variety of approaches that have shown strong performance in simulation [19], [20].

Real-world applications of RL for self-driving cars have been rarer, likely due to the difficulty in modeling the environment and specifying the reward function. An early notable example is [21], where they learn a steering policy for a real car. More recently, lane following was demonstrated using deep RL [22]. This approach controlled both speed and steering on a real car. We contrast the rural driving evaluations above with our experiments in busy Las Vegas.

Inverse reinforcement learning (IRL): IRL methods assume the expert is optimizing an unknown cost function that is learnable from demonstrations. An early application of IRL to self-driving cars was for parking lot navigation [23]. The method learned multiple different driving styles from a handful of demonstrations. However, the environment was static and the formulation assumed a linear combination of carefully handcrafted features.

Our approach is based on the popular maximum entropy formulation [24], which avoids ambiguities inherent in matching feature expectations. The maximum entropy IRL approach was extended to deep learning in [25], which avoided the need for laborious hand-engineering of features, and applied to simple benchmarks. The work of [26] is the most related to our approach, but their model only learned a dozen or so of feature (importance) weights and assumed a linear combination of handcrafted features. In addition, their method was only validated in simulation on a highway driving dataset.

III. Inverse Reinforcement Learning Planner

In this section, we describe our Inverse Reinforcement Learning (IRL) Planner as shown in Fig. 1. Our system consists of three stages: trajectory generation (Sec. III-B), an optional safety filtering step (Sec. III-C), and trajectory scoring (Sec. III-D). We rely on simple and reliable hand-engineered modules for trajectory generation and safety, and focus on learning how to score trajectories.

A. Input and output

Input: We encode the environment around our self-driving car using a mid-level representation. We assume that the ego is localized within a high-definition map and that objects are detected and tracked by a Perception system. Other road users (e.g., cars, bicyclists, and pedestrians) are represented by object type, an oriented bounding box, and speed. The high-definition map provides lane center-lines, road boundaries, traffic light locations, pedestrian crosswalks, speed limits, and other semantic information. We also provide a route, which indicates the lanes that the ego should traverse to make progress towards its goal.

We refer to the *scene context* at a given timestamp as a) the ego dynamic state $\mathcal{S}$ (oriented bounding box, speed), b) the other road users $\mathcal{U}$ (type, oriented bounding box, speed), c) the map $\mathcal{M}$, and d) the ego's desired route $\mathcal{R}$. The model receives the *scene context* at the current timestamp, a specified number of previous timestamps (e.g., the past $1\,\mathrm{s}$) as history $\mathcal{H}$ as well as world state forecasts $\mathcal{P}$ for a specified number of future timestamps (e.g., up to $6\,\mathrm{s}$) from a prediction module.

Output: Our planner generates multiple ego trajectories and scores each one according to how closely it matches the ground truth. A trajectory is a sequence of future states of the ego, where we assume that there is a fixed timestep between all states. Let $s_t = (x, y, \theta, v)$ represent a state at time t, with position (x, y), heading θ, and speed v. All values are with respect to a fixed coordinate frame defined at the ego's geometric center from the starting timestamp. The trajectory $\tau = s_1 \times s_2 \ldots \times s_t$, where t is the planned time horizon, that is ranked the best among a set of trajectories $\mathcal{T}$, is used as a reference for the vehicle's tracking and actuator controller.

B. Trajectory generation

The trajectory generation module uses the scene context to synthesize a diverse set of possible future motions for the ego. Important considerations for the ego's trajectory are that it a) is dynamically feasible and b) satisfies all requirements of the low-level tracking and actuator control

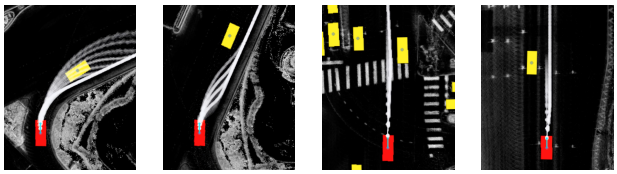


Fig. 2. Proposed trajectories for the ego (red rectangle). Each trajectory is shown in translucent white dot-line. Overlap is due to multiple acceleration profiles. All trajectories follow the route.

(i.e., levels of continuity, minimum turn radius, minimum acceleration from a stop). Secondary considerations are that the trajectory is compliant with the map (e.g., it stays on the road). While these considerations do not preclude using a learned trajectory generation module, we found that a simple hand-engineered trajectory generator can easily satisfy the considerations above.

The trajectory generator uses i) the current ego state $\mathcal{S}$, ii) the route $\mathcal{R}$, and iii) the map $\mathcal{M}$ to create a diverse set of ego trajectories $\mathcal{T}$, namely $(\mathcal{S}, \mathcal{R}, \mathcal{M}) \mapsto \mathcal{T}$. The generator integrates a desired acceleration profile along the route ahead of the ego. In our experiments, we specified a range of constant acceleration profiles ranging from a firm brake ($-5.0\ m/s^2$) to a moderate acceleration ($1.5\ m/s^2$). As the ego will not always be on the lane center-line (due to vehicle controller tracking errors), we smoothly connect the initial ego pose with the route with Dubins paths [11] where turning radii are a fixed set of parameters. In a typical scene, the trajectory generator usually creates 50-150 trajectories depending on the ego state and route. Some examples are shown in the Fig 2.

C. Safety filter

Our framework supports the application of an optional interpretable safety filter before the trajectory scoring step. The filtering is based on: 1) a set of conservative world assumptions used to predict the behavior of non-ego road users, 2) a set of trajectory modifiers which are applied to the ego trajectory, and 3) a set of safety checks which the modified ego trajectory needs to pass. The safety filter is similar in spirit to the fallback layer proposed by [17], except that 1) it directly filters the proposed trajectories, rather than projecting the output trajectory to a safe set, and 2) the trajectory modifier effectively implements a recursive safety guarantee: if we execute the first 1 s of a trajectory, can we remain safe by firmly decelerating after?

D. Trajectory scoring with maximum entropy IRL

Appropriately scoring trajectories is the core challenge of our planning approach. This difficulty arises because proper driving behavior is heavily influenced by the environment around us, including the behavior and goals of other road users, of which we only have a partial understanding.

Trajectories are scored by a deep neural network trained with a maximum entropy IRL loss [24]. We use expert demonstrations collected from a skilled human driving our vehicle. The loss favors trajectories that most closely match the expert demonstration $\tau^\star$ (in feature space). In particular, let $r(\tau)$ represent the reward of trajectory $\tau \in \mathcal{T}$. The probability of a trajectory τ^* being selected according to the maximum entropy principle is $P(\tau^\star) = \frac{\exp r(\tau^\star)}{\sum_{\tau \in \mathcal{T}} \exp r(\tau)}$.

The negative log-likelihood loss (NLL) on a dataset D is defined as $\ell(D) = -\sum_{d \in D} \log P(\tau^\star(d))$ where $\tau^\star(d)$ is the demonstrated trajectory on token $d \in D$. To address data imbalance issues, we augment NLL with focal loss [27] (using a γ of 2.0)

$$\ell(D) = -\sum_{d \in D} (1 - P(\tau^\star(d)))^\gamma \log P(\tau^\star(d)). \qquad (1)$$

Features: We compute features for each proposed trajectory to use as inputs to our neural network. These features can be based on any combination of a proposed trajectory τ, ego state $\mathcal{S}$, other road users $\mathcal{U}$, the map $\mathcal{M}$, route $\mathcal{R}$, the history $\mathcal{H}$, and the predictions $\mathcal{P}$, meaning that $F_i(\tau, \mathcal{S}, \mathcal{U}, \mathcal{M}, \mathcal{R}, \mathcal{H}, \mathcal{P}) \to f_i \in \mathbb{R}^{k_i}$, where F_i is the feature (extraction) function corresponding to feature i and k_i is its dimension. Below is a description of the features included in our base model (we have used $\|$ to denote the sequence concatenation operation).

- *Time-to-collision (TTC)*: the minimum number of seconds before the ego would collide with another road user in the (predicted) future. Specifically, $F_{TTC} : (\tau, \mathcal{S}, \mathcal{U}, \mathcal{M}, \mathcal{P}) \mapsto (f_1^{col}, \ldots, f_T^{col})$ where f_t^{col} denotes the minimum time to collision at timestamp t.
- *ACCInfo*: the ego speed, its distance to the road user ahead, the speed of the road user ahead, and the relative speed of the road user ahead. The feature function is $F_{ACCInfo} : (\tau, \mathcal{S}, \mathcal{U}, \mathcal{M}, \mathcal{P}) \mapsto f_1^{acc} \| \ldots \| f_T^{acc}$ with $f_t^{acc} \triangleq (d_{ahead}, b_{ahead}, v_{ego}, v_{ahead}, v_{ego} - v_{ahead})_t$ where d_{ahead} is the distance to the track ahead, b_{ahead} is a Boolean flag indicating if there is a track in front within a tunable distance (defaulted to 20 m), v_{ego} and v_{ahead} are the speed of the ego and the track ahead, respectively.
- *MaxJerk*: the maximum jerk (m/s^3) along the trajectory where $F_{MaxJerk} : (\tau, \mathcal{H}) \mapsto f^{jerk} \| (j_{max})$ where f^{jerk} is a thresholded Boolean encoding of the maximum jerk value j_{max} in the proposed trajectory.
- *MaxLateralAccel*: the max lateral acceleration (m/s^2) along the trajectory. $F_{MaxLateralAccel}$ is defined analogously to the *MaxJerk* feature above.
- *PastCoupling*: concatenation of the proposed and past ego poses in the local frame of reference, to encourage the model to maintain coherence between the past, present, and future states. Specifically, $F_{PastCoupling} : (\tau, \mathcal{H}) \mapsto f_1^{coup} \| \ldots \| f_T^{coup}$ where f_t^{coup} is the five-tuple $(x_{ego}, y_{ego}, \theta_{ego}, v_{ego}, a_{ego})_t$ denoting the ego state and acceleration at timestamp t.
- *SpeedLimit*: how closely the trajectory obeys the speed limit. $F_{SpeedLimit} : (\tau, \mathcal{S}, \mathcal{M}) \mapsto f_1^{limit} \| \ldots \| f_T^{limit}$ where $f_t^{limit} \triangleq (\frac{v_{ego} - v_{limit}}{v_{limit}}, b_{limit})_t$ with v_{limit} being the current speed limit and b_{limit} a Boolean flag that indicates whether the speed limit is exceeded.

Motion prediction: Decoupling prediction from planning to enable studying each system separately and to keep the presentation of DriveIRL simple and self-contained, we use an Intelligent Driver Model (IDM) [28] as our prediction model for other cars (with a conservative acceleration value to avoid assuming that stationary vehicles will speed up) and a constant velocity model for pedestrians and vehicles without a nearby lane.

Model architecture: To score a proposed trajectory, we adopt the architecture illustrated in Fig. 3 in which the extracted features are processed separately before interacting with one another through a masked self-attention mechanism. Specifically, each input feature f_i, as a temporal sequence of vehicle-environment interaction data, is first normalized via a BatchNorm1D layer before being fed to an LSTM module with 1 layer and a hidden size of 20. The output of the LSTM becomes the input to a feed-forward module followed by a self-attention mechanism with 2 heads and an embedding dimension of 120. Here we employ zero-masking of the queries to encode position. By taking into account other features through self-attention, the model produces for each feature a "corrected" output embedding that is converted to a scalar by a feed-forward network which is then $\tanh$ activated to produce a score $y_i \in (-1, 1)$. The final score for the proposal is the dot product of these feature scores and the corresponding feature importance weights w_i: $r(\tau) = \sum_i w_i y_i$. In total, our base (best) model has $\approx 88,700$ trainable parameters.

IV. Experiments

DriveIRL was evaluated on the dataset described in Sec IV-A. The metrics for comparison are explained in Sec IV-B. Various model ablation studies and a comparison with baselines are shown in Sec IV-C IV-D while simulation and real-world driving results are provided in Sec IV-E and IV-F.

A. Dataset

We created a self-driving car dataset that captures real-world urban driving in the center of Las Vegas. Our dataset is a part of the nuPlan [29] dataset that has been made public. It includes object annotations and high-definition maps. Vehicles, pedestrians, and bicyclists are automatically annotated using an offline perception system (similar in spirit to [30]) and viewed as ground truth. We performed filtering and extracted $182,032$ scenarios, each $11\,\mathrm{s}$ in duration ($1\,\mathrm{s}$ past, $10\,\mathrm{s}$ future), for a total of approximately $556\,\mathrm{h}$. Our main interest in deployment was to learn good adaptive cruise control (ACC) as well as turning (protected and unprotected) behaviors. Thus, we filtered out scenarios where the ego made lane changes or deviated far from the lane. After filtering, we performed a 3:1:1 split for train, val, and test sets. Tab. I shows a detailed distribution of our dataset by scenario tags.

TABLE I
A BREAKDOWN BY SCENARIO TAGS OF OUR DATASET OF $182,032$ $11\,\mathrm{s}$-SCENARIOS (ASV STANDS FOR APPROACHING STOPPED VEHICLE).

Tags	Straight	Right	Left	Stopped	Slow	Intersection	Close	ASV
Scenarios	163.1k	1.5k	2.6k	53.6k	37.4k	28.2k	11.2k	15.3k
Ratio	89.6 %	0.8 %	1.4 %	29.4 %	20.6 %	15.5 %	6.2 %	8.4 %

B. Metrics

We perform evaluation using a variety of metrics to give a full picture of driving. For reproducibility, we approximate the environment by a closed-loop replay for each $10\,\mathrm{s}$ scenario. From the initial scene context, we compute a planned ego trajectory, move along that trajectory for one step of $0.2\,\mathrm{s}$, replay the other agents, update the scene context, and repeat. Then, we compute metrics on the resulting executed trajectory as averages over the full $10\,\mathrm{s}$.

We have three high level metrics which are Boolean functions of "low level" ones. These are 1) Safety: a function of collision, off-road driving, tailgating and minimum time to collision, 2) Comfort: a function of acceleration, jerk and yaw rate, 3) Progress: a function of progress made along the route and deviation from it. We also compute an error metric based on a weighted sum of the ℓ_2 xy-error and the heading error averaged over the duration of the trajectory. The weight on the heading error is chosen to be 2.5, we refer to this metric as ℓ_2 with yaw. We note that since other vehicles do not react to the ego during the replay rollouts (e.g., if we drive slower than the expert in the data), the overall "Safety" score is really a lower bound on safety.

C. Model ablations

The ablation experiments below illustrate the importance of the chosen features, architecture, data augmentation, and the loss function in our model. In these experiments, we use a batch size of 64 and an Adam optimizer with an initial learning rate of 10^{-3}. Additionally, we use a "cosine annealing with warm restarts" scheduler, which gradually lowers the learning rate to a minimum of 10^{-4} and resets it every 7 epochs. All models are trained over 20 epochs on eight AWS g4dn-metal instances with eight 16 GB NVIDIA Tesla T4 GPUs each. Because closed-loop simulation and metrics evaluation are computationally expensive, we randomly sampled $1,000$ scenarios from our evaluation set for ablation studies and $3,000$ scenarios from the test set for the final performance evaluation against other baselines. Training and closed-loop metrics evaluation takes about an hour per epoch.

Feature importance: To understand the importance of each hand-engineered feature, an ablation study is conducted and summarized in Tab. II. The contribution of each feature is studied by dropping one of them out at a time. We observe that all features are important because the *Base* model which includes them receives the highest scores across all high-level metrics and has the lowest Collision rate. Even though the ℓ_2 error is a bit higher compared to *No MaxJerk*, the $0.089\,\mathrm{m}$ difference is not significant in the qualitative results. The results also demonstrate the importance of the *PastCoupling* feature in ensuring Comfort and show that the *TTC* feature significantly reduced the collision rate.

Data augmentation: As the reference trajectory is never followed perfectly by the vehicle, errors can accumulate. We

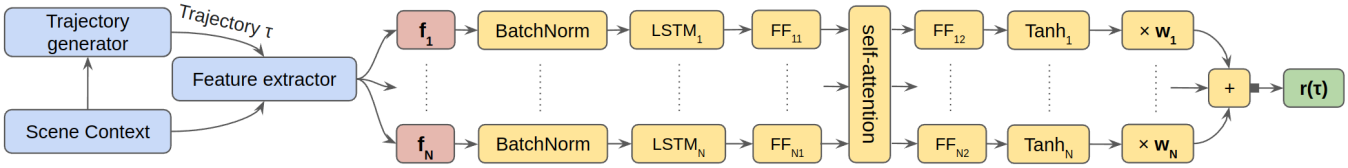


Fig. 3. Detailed trajectory scoring architecture.

TABLE II
ABLATION STUDY ON THE IMPORTANCE OF EACH FEATURE. THE ROW INDICATES THE FEATURE REMOVED FROM THE BASELINE MODEL. SEE SEC. III-D FOR DEFINITIONS.

Model	Safety	Comfort	Progress	ℓ_2 w/ yaw	Collision	Tailgate
Base (ours)	**0.925**	**0.840**	**0.988**	2.290	**0.001**	0.015
No TTC	0.865	0.790	0.958	2.679	0.055	0.070
No ACCInfo	0.862	0.817	0.959	2.832	0.004	0.035
No MaxJerk	0.863	0.825	0.960	**2.201**	**0.001**	0.029
No MaxLatAccel	0.917	0.821	0.982	2.299	0.005	**0.011**
No PastCoupling	0.901	0.697	0.979	2.905	0.005	**0.011**
No SpeedLimit	0.881	0.809	0.987	2.483	0.002	0.028

TABLE III
COMPARISON BETWEEN DIFFERENT AUGMENTATION SCHEMES.

Model	Safety	Comfort	Progress	ℓ_2 w/ yaw	Collision	Tailgate
Base (low noise)	**0.925**	0.840	**0.988**	2.290	**0.001**	**0.015**
No noise	0.850	**0.850**	0.956	**2.289**	0.005	0.055
High noise	0.917	0.845	0.986	2.525	0.002	0.016
Low past + present	0.921	0.817	0.982	2.302	0.005	0.019

augment the data by perturbing the ego's initial state during training to reduce sensitivity to such errors. For our low noise baseline, we use zero-mean Gaussian data augmentation for longitudinal offset ($1.2\,\mathrm{m}$ std), lateral offset ($0.8\,\mathrm{m}$ std), heading offset ($0.1\,\mathrm{rad}$ std), and velocity ($0.1\,\mathrm{m\,s^{-1}}$ std). For the high noise ablation, we respectively use $2.5\,\mathrm{m}$ std, $1.5\,\mathrm{m}$ std, $0.3\,\mathrm{rad}$ std, and $0.2\,\mathrm{m\,s^{-1}}$ std. The velocity is clamped to prevent it from being negative.

Architecture: We performed several ablations on the model architecture before settling on one described in Sec III-D. We show in Tab. IV that the other two extremes, namely, concatenating all input features and using them as one monolithic feature in a single feedforward network or siloing all input features (not allowing any interaction) both result in inferior performance. It may also be seen that input normalization and attention input masking are beneficial.

Loss: Tab. V shows that it is better to maximize the probability of the ℓ_2 projection of the ground truth onto the trajectory set instead of the ground truth itself. This is expected as the ground truth comes from a different distribution than the proposals in addition to being unavailable at inference time. Filtering possibly unsafe trajectories from the set before finding the ground truth projection is also crucial to obtaining a safe model. Doing the projection using the average ℓ_2 norm instead of an ℓ_2 norm with a yaw error penalty also seems preferable. Lastly from the same table, we can see that using focal loss as in Equation (1) improves performance. Further experiments comparing using focal loss against manually balancing datasets also favor former's effectiveness.

TABLE IV
COMPARISON BETWEEN DIFFERENT MODEL ARCHITECTURES.

Model	Safety	Comfort	Progress	ℓ_2 w/ yaw	Collision	Tailgate
Base (ours)	**0.925**	**0.840**	**0.988**	2.290	**0.001**	0.015
Monofeature FC	0.816	0.715	0.922	3.084	0.009	0.020
Siloed FC	0.900	0.784	0.964	2.394	0.018	0.021
No input norm	0.880	0.817	0.957	2.496	0.003	0.016
Attention no masking	0.910	0.835	0.983	**2.285**	0.004	**0.012**

TABLE V
COMPARISON BETWEEN DIFFERENT LOSS FUNCTIONS.

Metric	Safety	Comfort	Progress	ℓ_2 w/ yaw	Collision	Tailgate
Base (ours)	0.925	**0.840**	0.988	**2.290**	**0.001**	**0.015**
GT demo	0.910	0.553	**0.992**	2.870	0.012	0.029
Unsafe demo	0.873	0.823	0.977	2.518	0.036	0.049
Weighted yaw demo	**0.932**	0.839	0.984	2.530	0.005	0.018
Without focal loss	0.910	0.831	0.976	2.393	0.004	0.018

TABLE VI
BASELINES ON THE TEST SET. IDM = INTELLIGENT DRIVER MODEL. CS = CONSTANT SPEED.

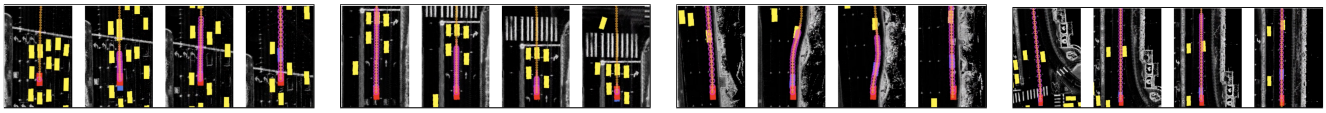
Metric	Safe	Comfort	Progress	ℓ_2 w/ yaw	Collision	Tailgate
Expert	1.000	0.984	1.000	0.000	0.000	0.000
Base (ours)	0.916	0.830	0.986	2.351	0.003	0.017
Base + Safety (ours)	**0.930**	0.815	0.971	**2.204**	**0.001**	**0.006**
IDM	0.891	0.898	**0.987**	4.478	0.005	0.019
CS lane follow	0.669	**0.992**	0.902	3.963	0.161	0.166

D. Baselines

We evaluate our model on a test dataset and compare it with an Intelligent Driver Model (IDM) [28] and a constant speed (CS) lane follow model. The IDM baseline is a well-known version of an expert planner that focuses on adaptive

TABLE VII
EVALUATION IN NUPLAN. IDM = INTELLIGENT DRIVER MODEL. SAFETYNET* = SAFETYNET [17] WITHOUT FALLBACK LAYER.

Metric	Score	Collision	Comfort	Progress	ℓ_2 w/ yaw	DrivableArea
Expert	0.965	0.995	0.960	1.000	0.00	0.992
Base (ours)	**0.803**	0.955	0.906	0.953	**8.67**	**0.992**
IDM	0.775	0.861	0.842	**0.999**	8.83	**0.992**
SafetyNet*	0.655	**0.986**	**0.974**	0.907	23.36	0.799



(a) Ego starting from a stop. (b) Ego stopping for the lead vehicle. (c) Adaptive cruise control. (d) A moderate cut-in.

Fig. 4. Qualitative driving performance in common scenarios.

Fig. 5. Smoothly stopping behind a vehicle in dense traffic on the Las Vegas Strip.

cruise control while the CS model is a simple lower-bound. The results are shown in Tab. VI. Our base model with the safety filter outperforms others in all safety related metrics. Without the safety filter, our base model still outperforms the IDM baseline, with a significantly lower ℓ_2 error, implying that our model resembles the human expert more. Furthermore, the base model also has higher scores in all safety related metrics in both high- and low-level scores like collision and tailgate rates.

As shown in Tab. VII, we also evaluate DriveIRL (with no safety filter) in the nuPlan framework against the IDM and SafetyNet*, which is an implementation of SafetyNet without the fallback layer. The evaluation is done on $1,000$ random 20-second Las Vegas test scenarios using closed-loop nonreactive agents. Our model surpasses both in the default nuPlan score (computed from multiple safety, comfort, and progress related metrics), weighted ℓ_2 error, and DrivableArea metric. SafetyNet* is found to be overly conservative in dense areas, as evidenced by its lower Progress score. It also tends to depart from lanes onto empty non-drivable areas, resulting in a high ℓ_2 error and a lower collision rate (high Collision score). In contrast, our model has high scores in all important metrics, consistent with its having the best overall score.

E. Simulation results

Fig. 4 exhibits some qualitative closed-loop simulation results of our planner driving in typical scenarios. These scenarios are shown as a sequence of snapshots over $10\,\mathrm{s}$ rollouts. The ego vehicle is shown as a red rectangle, the expert vehicle is in blue, and other vehicles are in yellow. The orange line is the planned route and the purple circles represent the planned trajectory for the next $6\,\mathrm{s}$.

F. Real-world driving

Prior to deploying on public roads, DriveIRL was rigorously tested in both simulation and private routes. The simulation tests consist of the same Las Vegas Strip route that was our deployment goal, and involve a high-fidelity model for the ego vehicle and numerous actors exhibiting a wide variety of behaviors. When deployed on the Strip, the vehicle was piloted by a vehicle operator who was trained to take over for unsafe behavior and situations outside of our operating domain, including construction zones, bus stops, and yielding for emergency vehicles. On the Strip, our planner handled heavy traffic, aggressive cut-ins, unpredictable drivers, and busy passenger pick-up/drop-off zones near the hotels and casinos. Without the safety filter, the vehicle remained in autonomous mode for 8.8 miles of the 11-mile route. Overrides occurred for mandatory takeover regions and twice for undesired behavior. With the safety filter, the vehicle remained in autonomous mode for 6.9 of 8.5 miles, with takeovers only occurring due to mandatory takeover regions. Fig. 5 shows a typical maneuver where our self-driving vehicle smoothly stops for a vehicle ahead while surrounded by multiple vehicles.

V. Conclusions

We introduced DriveIRL: the first planner, to our knowledge, to drive a car in dense, urban traffic using inverse reinforcement learning. By designing an architecture split into ego trajectory *generation*, *checking*, and *scoring*, we were able to leverage relatively easy and reliable methods of trajectory generation and safety checking. This architecture allowed the trajectory scoring component of our system to focus on learning important nuanced driving behavior in dense traffic. We demonstrated our planner on the busy Las Vegas Strip, where it showed strong real-world performance on challenging scenarios such as cut-ins, abrupt braking, and cluttered hotel pickup/dropoff zones. While this work is aimed at showcasing the strength of simple handcrafted features, our approach does not precluding mixing them with learned features, which along with a deeper architecture and a learnable trajectory set are fair topics for future research.

Acknowledgements

We would like to thank our colleagues Juraj Kabzan, Dimitris Geromichalos, Christopher Eriksen, Whye Kit Fong, Gordon Gustafson, Qiang Xu, Elena Corina Grigore, and Sunaya Bajracharya for their help and support throughout this project.

REFERENCES

[1] 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," *arXiv preprint arXiv:1903.11027*, 2019.

[2] P. Sun, H. Kretzschmar, X. Dotiwalla, A. Chouard, V. Patnaik, P. Tsui, J. Guo, Y. Zhou, Y. Chai, B. Caine, V. Vasudevan, W. Han, J. Ngiam, H. Zhao, A. Timofeev, S. Ettinger, M. Krivokon, A. Gao, A. Joshi, Y. Zhang, J. Shlens, Z. Chen, and D. Anguelov, "Scalability in perception for autonomous driving: Waymo Open Dataset," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[3] M.-F. Chang, J. Lambert, P. Sangkloy, J. Singh, S. Bak, A. Hartnett, D. Wang, P. Carr, S. Lucey, D. Ramanan, and J. Hays, "Argoverse: 3d tracking and forecasting with rich maps," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[4] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? The KITTI vision benchmark suite," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[5] 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*, 2018, pp. 4490–4499.

[6] 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/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 12 697–12 705.

[7] H. Cui, V. Radosavljevic, F. Chou, T. Lin, T. Nguyen, T. Huang, J. Schneider, and N. Djuric, "Multimodal trajectory predictions for autonomous driving using deep convolutional networks," in *International Conference on Robotics and Automation (ICRA)*, May 2019.

[8] Y. Chai, B. Sapp, M. Bansal, and D. Anguelov, "MultiPath: Multiple probabilistic anchor trajectory hypotheses for behavior prediction," in *3rd Conference on Robot Learning (CoRL)*, November 2019.

[9] 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*, 2020, pp. 14 074–14 083.

[10] 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.

[11] S. M. LaValle, *Planning Algorithms*. New York, NY, USA: Cambridge University Press, 2006.

[12] D. A. Pomerleau, "Alvinn: An autonomous land vehicle in a neural network," *Advances in neural information processing systems*, vol. 1, 1988.

[13] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba, "End to end learning for self-driving cars," 2016. [Online]. Available: https://arxiv.org/abs/1604.07316

[14] A. O. Mayank Bansal, Alex Krizhevsky, "ChauffeurNet: Learning to drive by imitating the best and synthesizing the worst," in *Robotics: Science and Systems (RSS)*, June 2019.

[15] O. Scheel, L. Bergamini, M. Wołczyk, B. Osiński, and P. Ondruska, "Urban driver: Learning to drive from real-world demonstrations using policy gradients," 2021. [Online]. Available: https://arxiv.org/abs/2109.13333

[16] W. Zeng, W. Luo, S. Suo, A. Sadat, B. Yang, S. Casas, and R. Urtasun, "End-to-end interpretable neural motion planner," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[17] M. Vitelli, Y. Chang, Y. Ye, M. Wołczyk, B. Osiński, M. Niendorf, H. Grimmett, Q. Huang, A. Jain, and P. Ondruska, "Safetynet: Safe planning for real-world self-driving vehicles using machine-learned policies," 2021. [Online]. Available: https://arxiv.org/abs/2109.13602

[18] 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.

[19] D. Chen, B. Zhou, V. Koltun, and P. Krähenbühl, "Learning by cheating," in *Conference on Robot Learning*. PMLR, 2020, pp. 66–75.

[20] D. Chen, V. Koltun, and P. Krähenbühl, "Learning to drive from a world on rails," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 15 590–15 599.

[21] M. Riedmiller, M. Montemerlo, and H. Dahlkamp, "Learning to drive a real car in 20 minutes," in *2007 Frontiers in the Convergence of Bioscience and Information Technologies*. IEEE, 2007, pp. 645–650.

[22] A. Kendall, J. Hawke, D. Janz, P. Mazur, D. Reda, J.-M. Allen, V.-D. Lam, A. Bewley, and A. Shah, "Learning to drive in a day," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 8248–8254.

[23] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," in *Proceedings of the Twenty-First International Conference on Machine Learning*, ser. ICML '04. New York, NY, USA: Association for Computing Machinery, 2004, p. 1. [Online]. Available: https://doi.org/10.1145/1015330.1015430

[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] 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*, pp. 1–13, 2021.

[27] 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.

[28] 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.

[29] H. Caesar, J. Kabzan, K. S. Tan, W. K. Fong, E. Wolff, A. Lang, L. Fletcher, O. Beijbom, and S. Omari, "nuplan: A closed-loop ml-based planning benchmark for autonomous vehicles," *arXiv preprint arXiv:2106.11810*, 2021.

[30] C. R. Qi, Y. Zhou, M. Najibi, P. Sun, K. Vo, B. Deng, and D. Anguelov, "Offboard 3d object detection from point cloud sequences," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2021, pp. 6134–6144.