Deep Imitative Models for Flexible Inference, Planning, and Control

Nicholas Rhinehart
Carnegie Mellon University
nrhineha@cs.cmu.edu

Rowan McAllister
UC Berkeley
rmcallister@berkeley.edu

Sergey Levine
UC Berkeley
svlevine@berkeley.edu

Abstract

Imitation learning is an appealing approach for autonomous control: in many tasks, demonstrations of preferred behavior can be readily obtained from human experts, removing the need for costly and potentially dangerous online data collection in the real world. However, imitation learning produces behavioral policies with limited flexibility to accommodate new goals at test-time. In contrast, model-based reinforcement learning (MBRL) can plan to arbitrary goals using a predictive dynamics model learned from data, yet suffers from two shortcomings. First, the dynamics alone cannot be used to choose desired or safe outcomes – it estimates only what is possible, not what is preferred. Second, MBRL typically requires additional online data collection to ensure the model is accurate in situations that are actually encountered when attempting to achieve test-time goals. Collecting this data with a partially-trained model can be dangerous and time-consuming. In this paper, we propose "imitative models" to combine the benefits of imitation learning and MBRL. Imitative models are probabilistic predictive models able to plan interpretable expert-like trajectories to achieve arbitrary goals. Inference with them resembles trajectory optimization in model-based reinforcement learning, and learning them resembles imitation learning. We find this method substantially outperforms six direct imitation learning approaches (five of them prior work) and an MBRL approach in a dynamic simulated autonomous driving task, and can be learned efficiently from a fixed set of expert demonstrations without additional online data collection. We also show our model can flexibly incorporate user-supplied costs at test-time, can plan to sequences of goals, and can even perform well with imprecise goals, including goals on the wrong side of the road.

1 Introduction

Reinforcement learning (RL) algorithms offer the promise of automatically learning desirable behaviors from raw sensory inputs with minimal engineering. However, RL generally requires *online* learning: the agent must collect more data with its latest strategy, use this data to update a model, and repeat. While this is natural in some settings, deploying a partially-trained policy on a real-world autonomous system, such as a car or robot, can be dangerous. In these settings, we argue learning behaviour should happen *offline* from expert demonstrations. How can we incorporate such demonstrations into a robotic system, like an autonomous car, to perform a variety of tasks? One option is imitation learning (IL), which can learn policies that stay near the expert's distribution. Another option is model-based methods [2, 5, 11], which can use the data to fit a dynamics model, and can in principle be used with planning algorithms to achieve any user-specified goal at test-time. However, in practice, model-based and model-free RL algorithms are vulnerable to distributional drift [22, 33]: when acting according to the learned model or policy, the agent visits states different from those seen during training, and in those it is unlikely to determine an effective course of action. This is especially problematic when the data intentionally excludes adverse events such as crashes. Therefore, MBRL algorithms usually require online collection and training [9, 16]. IL algorithms use expert demonstration data and, despite similar drift shortcomings [24], can sometimes learn effective

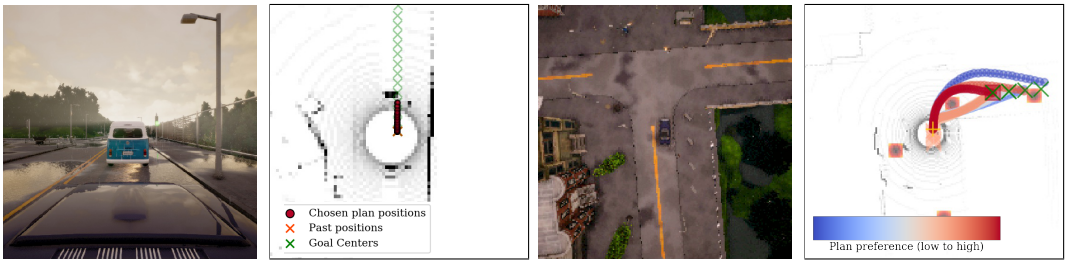

Figure 1: We apply our algorithm to navigation in CARLA [8]. *Left:* Image depicting the current scene, in which the light recently turned from green to red. *Left-Middle:* Plot showing LIDAR observations of our agent, the goals it received from a route planner, and the plan produced by our method. The model smoothly chooses between goals based on its prior of expert behavior. Here, the stationary agents chooses to accelerate to follow the vehicle ahead. *Right-Middle:* Image depicting an intersection scene. *Right:* LIDAR observations, goals, cost map of simulated potholes, and a variety of plans our method produces, colored by the planner's preference. Although the imitative model never observed pothole-avoidance behavior, it is able to plan a reasonable, on-road path around them with a test-time cost map. Its preferred plan enters the intersection and steers around a pothole.

policies without online data collection [36]. However, standard IL offers little task flexibility since it only predicts low-level behavior. While several works augmented IL with goal conditioning [3, 7], these goals *must be specified in advance during training*, and are typically simple (e.g., turning left or right). See Section 3 for further related work discussion.

Our goal is to devise an algorithm that combines the advantages of IL and MBRL by offering the flexibility to achieve new user-specified goals at test-time and the ability to learn entirely from offline data. By learning a deep conditional probabilistic forecasting model from expert data, we capture the distribution of expert behaviors without using manually designed reward functions. To plan to a goal, our method infers the most probable expert state trajectory under a posterior distribution induced by the model and a task-specifying goal distribution. By incorporating a model-based representation, our method can easily plan to previously unseen user-specified goals while behaving similar to the expert, and can be flexibly repurposed to perform a variety of test-time tasks without any additional training.

We demonstrate our method on a dynamic simulated autonomous driving task (see Fig. 1). A route planner provides navigational goals, which our joint deep probabilistic forecasting uses to automatically generate and follow interpretable plans that obey the rules of the road (e.g. stopping for vehicles, obeying traffic signals, driving in the correct lane). Videos are available at `https://sites.google.com/view/imitative-models`. Our approach's contributions are:

1. **Interpretable expert-like plans without reward engineering.** Our approach generates explicit multi-step expert-like plans, offering superior interpretability to one-step imitation learning models that *plan implicitly* to achieve goals. In contrast to MBRL, which can generate plans, our method generates expert-like behaviors without any reward function crafting.

2. **Flexibility to new tasks:** Our model can flexibly incorporate and achieve goals not seen during training, and can perform complex tasks that were never demonstrated, such as avoiding potholes.

3. **Robustness to goal specification noise:** We show that our approach is robust to noise in the provided task goals. In our application, we show that our agent can receive goals on the wrong side of the road, yet still navigate towards them while staying on the correct side of the road.

4. **Plan reliability estimation:** Our model can also be leveraged to test the *reliability* of a plan. We show that our model can discriminate between plans to real expert goals vs. plans towards other goals. This capability can be quite important for real-world safety-critical applications, such as autonomous driving, and can be used to build a degree of fault tolerance into the system.

5. **State-of-the-art CARLA performance:** Our approach substantially outperforms MBRL, a custom IL approach, and all five prior CARLA IL approaches known to us. It learned near-perfect driving through static and dynamic CARLA environments from expert observations alone.

2 Deep Imitative Models

We model continuous-state, discrete-time, partially-observed Markov processes. Our agent's state at time t is $\mathbf{s}_t \in \mathbb{R}^D$; $t = 0$ refers to the current time step. ϕ is the agent's observations. We distinguish variables in bold from functions (not bold). Random variables are capitalized. Absent subscripts denote *all* future time steps, e.g. $\mathbf{S} \doteq \mathbf{S}_{1:T} \in \mathbb{R}^{T \times D}$. We denote a probability density function of random variable $\mathbf{S}$ by $p(\mathbf{S})$, and the probability density at a specific value $\mathbf{s}$ as $p(\mathbf{s}) \doteq p(\mathbf{S}{=}\mathbf{s})$.

To learn agent dynamics that are possible *and* preferred, we construct a model of expert behavior. We fit a probabilistic model $q(\mathbf{S}|\phi)$ to expert trajectories $\mathcal{D} = \{\mathbf{s}^i, \phi^i\}_{i=1}^N$ drawn from an unknown distribution $\mathbf{s}^i \sim p(\mathbf{S}|\phi^i)$. By training $q(\mathbf{S}|\phi)$ to forecast expert trajectories with high likelihood, we model the scene-conditioned expert dynamics, which can score trajectories by how likely they are to come from the expert. A probabilistic model is necessary because expert behavior is stochastic and multimodal: e.g. choosing to turn either left or right at an intersection are both common decisions.

2.1 Imitative Planning to Goals

$q(\mathbf{S}|\phi)$ can generate trajectories that resemble those that the expert might take – e.g., we can generate trajectories that will navigate roads with human-like maneuvers and reasonable safety. However, these maneuvers will not have a specific goal. Besides generating human-like behaviors, we wish to *direct* our agent to desired goals at test-time and have the agent automatically reason about the necessary mid-level details. In general, we can define tasks by a set of goal variables $\mathcal{G}$. The probability density of a plan $\mathbf{s}^{\text{plan}}$ conditioned on the goal $\mathcal{G}$ is modelled by a posterior density $p(\mathbf{s}|\mathcal{G}, \phi)$. This posterior is implemented with a learned imitation prior $q(\mathbf{s}|\phi)$ and user-provided goal likelihood $p(\mathcal{G}|\mathbf{s}, \phi)$. We give examples of $p(\mathcal{G}|\mathbf{s}, \phi)$ after the planning derivation. The maximum a posteriori (MAP) inference procedure to generate expert-like plans that achieve abstract goals follows:

$$
\begin{aligned}
s^{\text{plan}} \doteq \arg\max_{\mathbf{s}} \mathcal{L}(\mathbf{s}, \mathcal{G}, \phi) &\doteq \arg\max_{\mathbf{s}} \ \log p(\mathbf{s}|\mathcal{G}, \phi) \\
&= \arg\max_{\mathbf{s}} \ \log q(\mathbf{s}|\phi) + \log p(\mathcal{G}|\mathbf{s}, \phi) - \log p(\mathcal{G}|\phi) \\
&= \arg\max_{\mathbf{s}} \ \log \underbrace{q(\mathbf{s}|\phi)}_{\text{imitation prior}} + \log \underbrace{p(\mathcal{G}|\mathbf{s}, \phi)}_{\text{goal likelihood}}. \qquad (1)
\end{aligned}
$$

We discuss gradient-ascent optimization of Eq. 1 in Appendix A and example goal likelihoods below.

Waypoint planning: One example of a concrete inference task is to plan towards a specific goal state, or waypoint. We can achieve this task by using a tightly-distributed goal likelihood function centered at the user's desired final state. This effectively treats a desired goal state, g_T, as if it were a noisy observation of a future state, with likelihood $p(\mathcal{G}|\mathbf{s}, \phi) = \mathcal{N}(g_T|\mathbf{s}_T, \epsilon I)$. The resulting inference corresponds to planning the trajectory $\mathbf{s}$ to a likely point under the distribution $\mathcal{N}(g_T|\mathbf{s}_T, \epsilon I)$. We can also plan to successive states with $\mathcal{G} = (g_{T-K}, \ldots, g_T)$ with goal likelihood $p(\mathcal{G}|\mathbf{s}, \phi) = \prod_{k=T-K}^{T} \mathcal{N}(g_k|\mathbf{s}_k, \epsilon I)$ if the user (or program) wishes to specify a desired end velocity or acceleration when reaching the final location g_T (Fig. 2). Alternatively, a planner may propose a set of waypoints with the intention that the agent should reach any one of them. This is possible by using a Gaussian mixture likelihood $p(\mathcal{G}|\mathbf{s}, \phi) = \frac{1}{K} \sum_{k=1}^{K} \mathcal{N}(g_k|\mathbf{s}_T, \epsilon I)$ and is useful if some of those final states are not reachable with an expert-like plan, as measured by $q(\mathbf{s}|\phi)$.

Waypoint planning leverages the advantage of goal-conditioned IL: a user or program can communicate *where* they desire the agent to go without knowing the best and safest actions. The planning-as-inference procedure produces paths similar to those an expert would have taken to reach the given goal. In contrast to black-box, model-free conditional imitation learning that predicts controls, our method produces an explicit multi-step plan, accompanied by an explicit score of the plan's quality. This provides both interpretability and an estimate of the feasibility of the plan.

Costed planning: If the user desires more control over the plan, our model has the additional flexibility to accept arbitrary user-specified costs c at test-time. For example, we may have updated knowledge of new hazards at test-time, such as a given map of potholes or a predicted cost map. Cost-based knowledge $c(\mathbf{s}_i|\phi)$ can be incorporated by including an optimality variable $\mathcal{C}$ in $\mathcal{G}$, where $p(\mathcal{C} = 1|\mathbf{s}, \phi) \propto \prod_{t=1}^{T} \exp -c(\mathbf{s}_t|\phi)$ [14, 34]. The goal log-likelihood is $\log p(\{g_T, \mathcal{C} = 1\}|\mathbf{s}, \phi) = \log \mathcal{N}(g_T|\mathbf{s}_T, \epsilon I) + \sum_{t=1}^{N} -c(\mathbf{s}_t|\phi)$. Examples of combining costed planning with a Gaussian-mixture goal likelihood are shown in Fig. 3.

Figure 2: Imitative planning to goals: multi-goal waypoint planning enables fine-grained control of the plans.

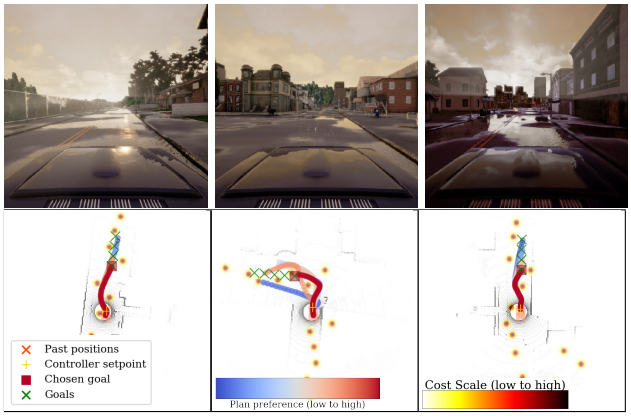

Figure 3: Costs can be assigned to "potholes" only seen at test-time; expert demonstrations with potholes were never observed. The planner prefers routes around the potholes.

2.2 Problem Setup

We model our agent's state at time t as $\mathbf{s}_t \in \mathbb{R}^D$ with $D=2$ in our application, where $\mathbf{s}_t$ represents our agent's (x, y) coordinates on the ground plane. The agent has access to environment perception $\phi \doteq \{\mathbf{s}_{-\tau:0}, \boldsymbol{\chi}, \boldsymbol{\lambda}\}$, where τ is the number of past positions we condition on, $\boldsymbol{\chi}$ is a high-dimensional observation of the scene, and $\boldsymbol{\lambda}$ is a low-dimensional traffic light signal. $\boldsymbol{\chi}$ could represent either LIDAR or camera images (or both), and is the agent's observation of the world. In our setting, we featurize LIDAR to $\boldsymbol{\chi} = \mathbb{R}^{200\times200\times2}$, with $\boldsymbol{\chi}_{ij}$ representing a 2-bin histogram of points above and at ground level in a 0.5m^2 cell at position (i, j). The CARLA simulator provides $\mathbf{s}_{-\tau:0}$ and $\boldsymbol{\lambda}$.

A deep imitative model forecasts future expert behavior. Its primary structural requirement is the ability to compute $q(\mathbf{s}|\phi)$. The ability to also compute gradients $\nabla_{\mathbf{s}} q(\mathbf{s}|\phi)$ enables gradient-based optimization for planning. A variety of prior work has focused on forecasting the behavior of other vehicles or pedestrians [25], which includes generative models based on some context inputs such as LIDAR, images, or known positions of external agents [10, 13, 17, 27, 37]. However, these methods cannot evaluate trajectory probabilities or repurpose their models to perform other inference tasks. A model that can compute probabilities is the Reparameterized Pushforward Policy (R2P2) [21], an invertible generative model [6, 20]. We extend R2P2 to instantiate the deep imitative model $q(\mathbf{S}|\phi)$. R2P2 was previously used to forecast vehicle trajectories, instead of to plan and control a vehicle.

In R2P2, $q_\theta(\mathbf{S}|\phi)$ is induced by an invertible, differentiable function: $\mathbf{S} = f_\theta(\mathbf{Z}; \phi) : \mathbb{R}^{T\times2} \mapsto \mathbb{R}^{T\times2}$; f_θ warps latent samples from a base distribution $\mathbf{Z} \sim q_0 = \mathcal{N}(0, I)$ to the output space over $\mathbf{S}$. θ is trained to maximize $q_\theta(\mathbf{S}|\phi)$ of expert trajectories. f_θ is defined for $1..T$ as follows:

$$\mathbf{S}_t = f_t(\mathbf{Z}_{1:t}) = \mu_\theta(\mathbf{S}_{1:t-1}, \phi) + \sigma_\theta(\mathbf{S}_{1:t-1}, \phi)\mathbf{Z}_t, \tag{2}$$

where $\mu_\theta(\mathbf{S}_{1:t-1}, \phi) = 2\mathbf{S}_{t-1} - \mathbf{S}_{t-2} + m_\theta(\mathbf{S}_{1:t-1}, \phi)$ encodes a constant-velocity inductive bias. The $m_\theta \in \mathbb{R}^2$ and $\sigma_\theta \in \mathbb{R}^{2\times2}$, which represent reactive mean and covariance in acceleration, are computed by expressive, nonlinear neural networks. In our application, they observe previous states, LIDAR input $\boldsymbol{\chi}$, and traffic-light information $\boldsymbol{\lambda}$. The resulting trajectory distribution is complex and multimodal. Because the state of traffic lights cannot be inferred from LIDAR alone, we extended

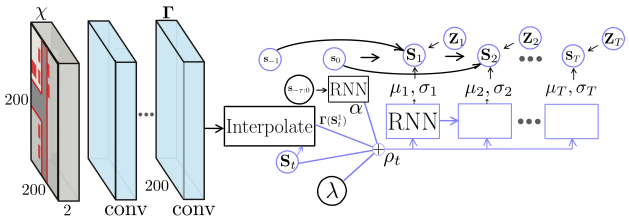

Figure 4: Architecture of m_θ and σ_θ, which parameterize $q(\mathbf{S}|\phi)$. See Appendix C for details.

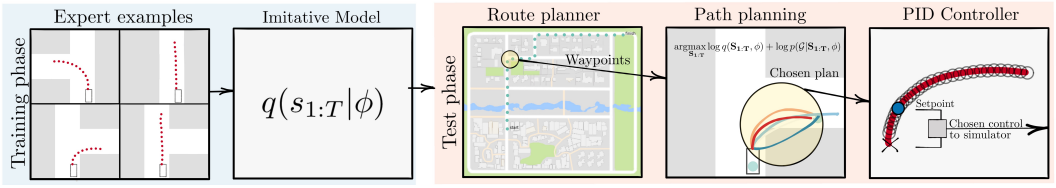

Figure 5: Illustration of our method applied to autonomous driving. Our method trains an imitative model from a dataset of expert examples. After training, the model is repurposed as an imitative planner. At test-time, a route planner provides waypoints to the imitative planner, which computes expert-like paths to each goal. The best plan is chosen according to the planning objective and provided to a low-level PID-controller in order to produce steering and throttle actions.

the "RNN" architecture described in R2P2 to observe $\boldsymbol{\lambda}$. We found this helpful to fixing cases where $q(\mathbf{S}|\phi)$ exhibited no preference to moving forward when the agent was already stopped, and helpful to improving $q(\mathbf{S}|\phi)$'s preference of stopping at red lights. We used $T=40$ trajectories at 10Hz (4 seconds of prediction or planning), $\tau=20$. Fig. 4 depicts the architecture of μ_θ and σ_θ.

2.3 Imitative Driving

Having described the imitative models method, we will now instantiate a complete autonomous driving framework based on imitative models to study in our experiments. We use three layers of spatial abstraction to plan to a faraway destination, common to model-based (not end-to-end) autonomous vehicle setups: coarse route planning over a road map, path planning within the observable space, and feedback control to follow the planned path [19, 28]. For instance, a route planner based on a conventional GPS-based navigation system might output waypoints roughly in the lanes of the desired direction of travel, but not accounting for environmental factors such as the positions of other vehicles. This communicates to the path planner *where* the vehicle should go, but not *when* or *how* it should get there. The waypoints are treated as goals and passed to the imitative planner, which generates a path chosen according to the optimization in Eq. 1. We found that the Gaussian mixture likelihood worked quite well for this setting, which allows the imitative planner to roughly choose which of K waypoints it prefers according to $L(\mathbf{s}, \mathcal{G}, \phi) = q(\mathbf{s}|\phi) + \frac{1}{K}\sum_{k=1}^{K}\mathcal{N}(g_k|\mathbf{s}_T, \epsilon I)$. We observed the planner prefers *closer* waypoints when obstructing vehicles were observed in the LIDAR, when the vehicle was already stopped, and when a red light was detected; we observed the planner prefers *farther* waypoints that maintained forward velocity when there were no obstructing vehicles and when green lights or no lights were observed. These sophisticated behaviors are a result of $q(\mathbf{S}|\phi)$ accurately internalizing expert demonstrations. The resulting plans are fed to a low-level controller (we use a PID-controller) that follows the plan. In Fig. 5 we illustrate how we use our model in our application. Further details of this algorithm are given in Appendix A.

3 Related Work

Several prior works [1, 29, 32] used imitation learning to train policies that contain planning-like modules as part of the model architecture. While our work also combines planning and imitation learning, the goal is instead to capture a distribution over possible trajectories, and then plan trajectories at test-time that accomplish a variety of given goals with high probability under this distribution.

A body of previous work has explored conditional IL for autonomous driving in the CARLA simulator [3, 4, 15, 16, 26]. Model-free Behavior Cloning (BC) approaches [3, 4, 15, 26] condition on scene images and a discrete set of directives by a high-level planner. [16] is a BC+RL approach that collects data online to bootstrap a policy learned from observations. While model-free conditional IL can be effective given a discrete set of user directives, our model-based conditional IL has several advantages: flexibility to handle more complex directives post-training (e.g. avoiding hazardous potholes (Fig. 3), the ability to rank plans and goals, and can generate explicit interpretable planned and unplanned trajectories. Table 1 compares attributes of each method.

To our knowledge, no Model-Based Reinforcement Learning approach has been applied to the CARLA simulator. However, MBRL is related to our method because it also learns a predictive model of dynamics (albiet a one-step model $\mathbf{s}_{t+1} = \hat{f}(\mathbf{s}_{t-3:t}, \mathbf{a}_t)$). MBRL plans through this model in order to optimize a reward function. Because the dynamics of MBRL captures only what is

possible, rather than what is expert-preferred, the task of evoking expert-like behavior is offloaded to the reward function, which can be difficult and time-consuming to craft properly. We devised an MBRL approach that uses the same LIDAR map χ our method uses to locate obstacles, in addition to extra information about the past positions of other agents that our model *does not* use. It uses this information to score plans with rewards, and uses its dynamics model to plan a reachability tree [12] to the same waypoints our method and CILWP receive. The "CILWP" approach is based on [3], except it uses visual inputs and waypoint inputs identical to ours. We employ a network model architecture nearly identical to that of our main approach. Setpoints are predicted instead of actions and then passed to a PID controller, which means CILWP shares the same low-level and high-level controllers as our approach. See Appendix D for further details of the baselines we implemented.

Table 1: Approach attributes. † indicates methods we have implemented. * indicates results reported in [4]. "BC" stands for Behavior Cloning: these approaches predict the next one-step action.

Attribute	CIRL* [16]	CAL*[26]	MT* [15]	CIL* [3]	CILRS*[4]	CILWP†	MBRL† [12]	*Ours*†
Low controller	–	PID	–	–	–	PID	–	PID
Mid controller	BC+RL	BC	BC	BC	BC	BC-Setpoints	Reach. Tree	IM
High controller	Command	Command	Command	Command	Command	Waypoint	Waypoint	Waypoint
LIDAR input	✗	✗	✗	✗	✗	✓	✓	✓
Image input	✓	✓	✓	✓	✓	✗	✗	✗
Traffic light input	✗	✗	✗	✗	✗	✓	✗	✓
Model-based	✗	✗	✗	✗	✗	✗	✓	✓
Train Offline	✗	✓	✓	✓	✓	✓	✓	✓
Imitative	✓	✗	✓	✓	✓	✓	✗	✓
Flexible Reward	✗	✗	✗	✗	✗	✗	✓	✓
Expert PDF	✗	✗	✗	✗	✗	✗	✗	✓

Our approach is well-suited to offline settings where interactively collecting data is infeasable (e.g. if making mistakes is costly). However, *online* imitation learning is an active area of research in the case of hybrid IL-RL [16, 23, 31], and safe IL [18, 30, 35].

4 Experiments

We evaluate our method using the CARLA urban driving simulator [8]. We begin by training $q(\mathbf{S}|\phi)$ on a dataset of 25 hours of driving we collected in `Town01`, detailed in Appendix C.1. Following existing protocol, each test episode begins with the vehicle randomly positioned on a road in the `Town01` or `Town02` maps in one of two settings: static-world (no other vehicles) or dynamic-world (with other vehicles). Following existing protocol, 25 episodes of behavior are collected in each of these four scenarios in which the agent must navigate to faraway destinations. We did our best to ensure faithful comparison to previous evaluations, but there are a variety of inevitable differences between the approaches, as shown in Table 1. The most significant difference is the form of the high-level controller. Our method shares the exact same high-level controller with the baselines we implemented: our waypointer applies A* search to CARLA's road-graph to plan a route from the start location to the destination. In contrast, the prior CARLA work assumes access to high-level *commands* at training and test time: "follow the lane (default), drive straight at the next intersection, turn left at the next intersection, and turn right at the next intersection" [8]. These commands are similar to the information a person might receive from a map-based navigational system. Waypoints are a reasonable assumption in settings with known maps or lane information. As we show later, even if these waypoints are imprecise (e.g. noisy or in the wrong lane), our method still performs well. In prior work, the command set and labels of the commands *must be specified before training*, whereas our model uses *no high-level command labels*, only histories of observations and positions.

Metrics and Results. We use three metrics: 1) success rate in driving to the goal location without any collisions (which all prior work reports); 2) red-light violations; and 3) proportion of time spent driving in the wrong lane and off road. With the exception of the "Success" metric, lower numbers are better. Performance results that compare our methods against baselines and prior work according to these metrics are shown in Table 2. We observe our method to outperform all other approaches in all settings: static world, dynamic world, training conditions, and test conditions.

4.1 Noise Robustness and Reliability Estimation

While our method makes use of a high-level planner to output potential waypoints, its performance is not entirely reliant on the quality of these waypoints. In this section, we analyze the performance of this method when the path planner is heavily degraded, to understand its stability and reliability.

Table 2: We evaluate different autonomous driving methods on CARLA's *Navigation* and *Navigation Dynamic* tasks. A † indicates methods we have implemented. A * indicates results reported in [4]. A "–" indicates an unreported statistic. A ‡ indicates an optimistic estimate in transferring a result from the static setting to the dynamic setting. "*Our method*" uses a Gaussian mixture waypointer; "*Our method S.*" uses a "smart" waypointer reactive to light state, detailed in Appendix B.1.

		`Town01` (training conditions)				`Town02` (test conditions)			
Static Nav. Method		Success	Ran Red Light	Wrong lane	Off road	Success	Ran Red Light	Wrong lane	Off road
CIRL*	[16]	93%	–	–	–	68%	–	–	–
CAL*	[26]	92%	–	–	–	68%	–	–	–
MT*	[15]	81%	–	–	–	78%	–	–	–
CIL*	[3]	86%	83%	–	–	44%	82%	–	–
CILRS*	[4]	95%	27%	–	–	90%	64%	–	–
CILWP†		28%	**0.0%**	0.38%	10.23%	36%	**0.0%**	1.69%	16.82%
MBRL†	[12]	**96%**	78%	14.3%	1.94%	**96%**	73%	19.6 %	0.75%
Our method†		**96%**	0.83%	**0.01%**	0.08%	**96%**	**0.0%**	**0.03%**	**0.14%**
Our method S.†		**96%**	**0.0%**	0.04%	**0.07%**	92%	**0.0%**	0.18%	0.27%
Dynamic Nav. Method		Success	Ran Red Light	Wrong lane	Off road	Success	Ran Red Light	Wrong lane	Off road
CIRL*	[16]	82%	–	–	–	41%	–	–	–
CAL*	[26]	83%	–	–	–	64%	–	–	–
MT*	[15]	81%	–	–	–	53%	–	–	–
CIL*	[3]	83%	83%‡	–	–	38%	82%‡	–	–
CILRS*	[4]	92%	27%‡	–	–	66%	64%‡	–	–
CILWP†		17%	**0.0%**	0.20%	12.1%	36%	**0.0%**	1.11%	11.70%
MBRL†	[12]	64%	72%	11.1%	2.96%	48%	54%	20.6%	13.3 %
Our method†		92%	6.3%	0.04%	**0.005%**	**100%**	12%	0.11%	**0.04%**
Our method S.†		**100%**	1.7%	**0.03%**	**0.005%**	92%	**0.0%**	**0.05%**	0.15%

Navigating with high-variance waypoints. As a test of our model's capability to stay in the distribution of demonstrated behavior, we designed a "decoy waypoints" experiment, in which *half* of the waypoints are highly perturbed versions of the other half, serving as distractions for our planner. The planner is tasked with planning to all of the waypoints under the Gaussian mixture likelihood. We observed the imitative model to be surprisingly robust to decoy waypoints. Examples of this robustness are shown in Fig. 6. In Table 3, we report the success rate and the mean number of planning rounds for failed episodes in the "½ distractors" row. These numbers indicate our method can execute dozens of planning rounds without decoy waypoints causing a catastrophic failure, and often it can execute the hundreds necessary to achieve the goal. See Appendix E for details.

Navigating with waypoints on the wrong side of the road. We also designed an experiment to test our method under systemic bias in the route planner. Our method is provided waypoints on the wrong side of the road (in CARLA, the left side), and tasked with following the directions of these waypoints while staying on the *correct* side of the road (the right side). In order for the value of $q(\mathbf{s}|\phi)$ to outweigh the influence of these waypoints, we increased ϵ of $p(\mathcal{G}|\mathbf{s},\phi) = \frac{1}{K}\sum_{k=1}^{K}\mathcal{N}(g_k|\mathbf{s}_T, \epsilon I)$ by tuning it on the simulator. We found our method to still be very effective at navigating, and report

Figure 6: Tolerating bad waypoints. The planner prefers waypoints in the distribution of expert behavior (on the road at a reasonable distance). *Columns 1,2*: Planning with ½ decoy waypoints. *Columns 3,4*: Planning with all waypoints on the wrong side of the road.

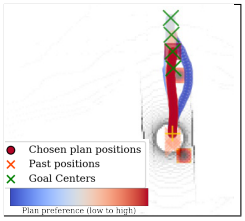

Figure 7: Test-time plans prefer steering around potholes.

Table 3: **Robustness to waypoint noise and test-time pothole adaptation**. Our method is robust to waypoints on the wrong side of the road, and fairly robust to decoy waypoints. Our method is flexible enough to safely produce behavior not demonstrated (pothole avoidance) by incorporating a test-time cost. Ten episodes are collected in each Town.

		`Town01` (training conditions)			`Town02` (test conditions)		
Waypointer	Extra Cost	Success	Wrong lane	Potholes hit	Success	Wrong lane	Potholes hit
Noiseless waypointer		100%	0.00%	177/230	100%	0.41%	82/154
Waypoints wrong lane		100%	0.34%	–	70%	3.16%	–
$1/2$ waypoints distracting		70%	–		50%	–	–
Noiseless waypointer	Pothole	90%	1.53%	10/230	70%	1.53%	35/154

results in Table 3. This further illustrates our method's tendency to stay near the distribution of expert behavior, as our expert never drove on the wrong side of the road.

Plan reliability estimation. Besides using our model to make a best-effort attempt to reach a user-specified goal, the fact that our model produces explicit likelihoods can also be leveraged to test the *reliability* of a plan by evaluating whether reaching particular waypoints will result in human-like behavior or not. This capability can be quite important for real-world safety-critical applications, such as autonomous driving, and can be used to build a degree of fault tolerance into the system. We designed a classification experiment to evaluate how well our model can recognize safe and unsafe plans. We planned our model to known good waypoints (where the expert actually went) and known bad waypoints (off-road) on 1650 held-out test scenes. We used the planning criterion to classify these as good and bad plans and found that we can detect these bad plans with 97.5% recall and 90.2% precision. This result indicates imitative models could be effective in estimating the reliability of plans. Further details are in Appendix E.2.

4.2 Producing Unobserved Behaviors to Avoid Novel Obstacles

To further illustrate our model's capability to incorporate test-time costs, we designed a pothole collision experiment. We simulated potholes in the environment by randomly inserting them in the cost map near waypoints. We ran our method with a test-time-only cost map of the simulated potholes, and compared to our method that did not incorporate the cost map (and thus had no incentive to avoid potholes). We recorded the number of collisions with potholes. In Table 3, our method with cost incorporated avoided most potholes while avoiding collisions with the environment. To do so, it drove closer to the centerline, and occasionally dipped into the opposite lane. Our model internalized obstacle avoidance by staying on the road, and demonstrated its flexibility to obstacles not observed during training. Fig. 7 shows an example of this behavior. See Appendix F for details.

5 Discussion

We proposed a method that combines elements of imitation learning and model-based reinforcement learning (MBRL). Our method first learns preferred behavior by estimating the distribution of expert demonstrations, and then plans paths to achieve user-specified goals at test-time while maintaining high probability under the expert distribution. We demonstrated several advantages and applications of our algorithm in autonomous driving scenarios. In the context of MBRL, our method mitigates the distributional drift issue by explicitly preferring plans that stay close to the expert demonstration data. This implicitly encourages safety: in contrast to MBRL, which requires negative examples to understand the potential for adverse outcomes (e.g., crashes), our method tends to avoid these outcomes because they do not occur (or rarely occur) in the training data. In the context of imitation learning, our method provides a flexible, safe way to generalize to new goals by planning, compared to prior work on black-box, model-free conditional imitation learning. Our algorithm produces an explicit plan within the distribution of preferred behavior accompanied with a score: the former offers interpretability, and the latter provides an estimate of the feasibility of the plan. We believe our method is broadly applicable in settings where expert demonstrations are available, flexibility to new situations is demanded, and safety is paramount.

References

[1] Brandon Amos, Ivan Dario Jimenez Rodriguez, Jacob Sacks, Byron Boots, and Zico Kolter. Differentiable MPC for end-to-end planning and control. 2018.

[2] Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Advances in Neural Information Processing Systems*, pages 4754–4765, 2018.

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

[4] Felipe Codevilla, Eder Santana, Antonio M López, and Adrien Gaidon. Exploring the limitations of behavior cloning for autonomous driving. *arXiv preprint arXiv:1904.08980*, 2019.

[5] Marc Deisenroth and Carl E Rasmussen. PILCO: A model-based and data-efficient approach to policy search. In *International Conference on Machine Learning (ICML)*, pages 465–472, 2011.

[6] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using Real NVP. *arXiv preprint arXiv:1605.08803*, 2016.

[7] Alexey Dosovitskiy and Vladlen Koltun. Learning to act by predicting the future. *arXiv preprint arXiv:1611.01779*, 2016.

[8] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *Conference on Robot Learning (CoRL)*, pages 1–16, 2017.

[9] Peter Englert, Alexandros Paraschos, Marc Peter Deisenroth, and Jan Peters. Probabilistic model-based imitation learning. *Adaptive Behavior*, 21(5):388–403, 2013.

[10] Agrim Gupta, Justin Johnson, Li Fei-Fei, Silvio Savarese, and Alexandre Alahi. Social GAN: Socially acceptable trajectories with generative adversarial networks. In *Computer Vision and Pattern Recognition (CVPR)*, number CONF, 2018.

[11] Leonid Kuvayev and Richard S. Sutton. Model-based reinforcement learning with an approximate, learned model. In *Yale Workshop on Adaptive and Learning Systems*, pages 101–105, 1996.

[12] Steven M LaValle. Planning algorithms. chapter 14, pages 802–805. Cambridge University Press, 2006.

[13] Namhoon Lee, Wongun Choi, Paul Vernaza, Christopher B Choy, Philip HS Torr, and Manmohan Chandraker. DESIRE: Distant future prediction in dynamic scenes with interacting agents. In *Computer Vision and Pattern Recognition (CVPR)*, pages 336–345, 2017.

[14] Sergey Levine. Reinforcement learning and control as probabilistic inference: Tutorial and review. *arXiv preprint arXiv:1805.00909*, 2018.

[15] Zhihao Li, Toshiyuki Motoyoshi, Kazuma Sasaki, Tetsuya Ogata, and Shigeki Sugano. Rethinking self-driving: Multi-task knowledge for better generalization and accident explanation ability. *arXiv preprint arXiv:1809.11100*, 2018.

[16] Xiaodan Liang, Tairui Wang, Luona Yang, and Eric Xing. CIRL: Controllable imitative reinforcement learning for vision-based self-driving. *arXiv preprint arXiv:1807.03776*, 2018.

[17] Wei-Chiu Ma, De-An Huang, Namhoon Lee, and Kris M Kitani. Forecasting interactive dynamics of pedestrians with fictitious play. In *Computer Vision and Pattern Recognition (CVPR)*, pages 4636–4644. IEEE, 2017.

[18] Kunal Menda, Katherine Driggs-Campbell, and Mykel J Kochenderfer. DropoutDAgger: A Bayesian approach to safe imitation learning. *arXiv preprint arXiv:1709.06166*, 2017.

[19] Brian Paden, Michal Čáp, Sze Zheng Yong, Dmitry Yershov, and Emilio Frazzoli. A survey of motion planning and control techniques for self-driving urban vehicles. *Transactions on Intelligent Vehicles*, 1(1):33–55, 2016.

[20] Danilo Jimenez Rezende and Shakir Mohamed. Variational inference with normalizing flows. *arXiv preprint arXiv:1505.05770*, 2015.

[21] Nicholas Rhinehart, Kris M. Kitani, and Paul Vernaza. R2P2: A reparameterized pushforward policy for diverse, precise generative path forecasting. In *European Conference on Computer Vision (ECCV)*, September 2018.

[22] Stéphane Ross and Drew Bagnell. Efficient reductions for imitation learning. In *International Conference on Artificial Intelligence and Statistics*, pages 661–668, 2010.

[23] Stephane Ross and J Andrew Bagnell. Reinforcement and imitation learning via interactive no-regret learning. *arXiv preprint arXiv:1406.5979*, 2014.

[24] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *International Conference on Artificial Intelligence and Statistics*, pages 627–635, 2011.

[25] Andrey Rudenko, Luigi Palmieri, Michael Herman, Kris M Kitani, Dariu M Gavrila, and Kai O Arras. Human motion trajectory prediction: A survey. *arXiv preprint arXiv:1905.06113*, 2019.

[26] Axel Sauer, Nikolay Savinov, and Andreas Geiger. Conditional affordance learning for driving in urban environments. *arXiv preprint arXiv:1806.06498*, 2018.

[27] Edward Schmerling, Karen Leung, Wolf Vollprecht, and Marco Pavone. Multimodal probabilistic model-based planning for human-robot interaction. In *International Conference on Robotics and Automation (ICRA)*, pages 1–9. IEEE, 2018.

[28] Wilko Schwarting, Javier Alonso-Mora, and Daniela Rus. Planning and decision-making for autonomous vehicles. *Annual Review of Control, Robotics, and Autonomous Systems*, 1: 187–210, 2018.

[29] Aravind Srinivas, Allan Jabri, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Universal planning networks: Learning generalizable representations for visuomotor control. In *International Conference on Machine Learning*, pages 4739–4748, 2018.

[30] Liting Sun, Cheng Peng, Wei Zhan, and Masayoshi Tomizuka. A fast integrated planning and control framework for autonomous driving via imitation learning. *arXiv preprint arXiv:1707.02515*, 2017.

[31] Wen Sun, James Andrew Bagnell, and Byron Boots. Truncated horizon policy search: Combining reinforcement learning and imitation learning. In *International Conference on Learning Representations (ICLR)*, 2018.

[32] Aviv Tamar, Yi Wu, Garrett Thomas, Sergey Levine, and Pieter Abbeel. Value iteration networks. In *Advances in Neural Information Processing Systems*, pages 2154–2162, 2016.

[33] Sebastian Thrun. Learning to play the game of chess. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1069–1076, 1995.

[34] Emanuel Todorov. Linearly-solvable Markov decision problems. In *Advances in neural information processing systems*, pages 1369–1376, 2007.

[35] Jiakai Zhang and Kyunghyun Cho. Query-efficient imitation learning for end-to-end simulated driving. In *AAAI*, pages 2891–2897, 2017.

[36] Tianhao Zhang, Zoe McCarthy, Owen Jowl, Dennis Lee, Xi Chen, Ken Goldberg, and Pieter Abbeel. Deep imitation learning for complex manipulation tasks from virtual reality teleoperation. In *International Conference on Robotics and Automation (ICRA)*, pages 1–8. IEEE, 2018.

[37] Alex Zyner, Stewart Worrall, and Eduardo Nebot. Naturalistic driver intention and path prediction using recurrent neural networks. *arXiv preprint arXiv:1807.09995*, 2018.

A Algorithms

Algorithm 1 IMITATIVEDRIVING(ROUTEPLAN, IMITATIVEPLAN, PIDCONTROLLER, q_θ, H)

$\phi \leftarrow$ ENVIRONMENT($\emptyset$) {Initialize the robot}
while not at destination **do**
  $\mathcal{G} \leftarrow$ ROUTEPLAN(ϕ) {Generate waypoints}
  $\mathbf{s}_{1:T}^{\text{plan}} \leftarrow$ IMITATIVEPLANR2P2($q_\theta, \mathcal{G}, \phi$) {Plan path}
  for $h = 0$ to H **do**
    $u \leftarrow$ PIDCONTROLLER($\phi, \mathbf{s}_{1:T}^{\text{plan}}, h$)
    $\phi \leftarrow$ ENVIRONMENT(u) {Execute control}
  end for
end while

In Algorithm 1, we provide pseudocode for receding-horizon control via our imitative model. In Algorithm 2 we provide pesudocode that describes how we plan in the latent space of the trajectory. Since $\mathbf{s}_{1:T} = f(\mathbf{z}_{1:T})$ in our implementation, and f is differentiable, we can perform gradient descent of the same objective in terms of $\mathbf{z}_{1:T}$. Since q is trained with $\mathbf{z}_{1:T} \sim \mathcal{N}(0, I)$, the latent space is likelier to be better numerically conditioned than the space of $\mathbf{s}_{1:T}$, although we did not compare the two approaches formally.

Algorithm 2 IMITATIVEPLANR2P2($q_\theta, \mathcal{G}, \phi, f$)

Define MAP objective $\mathcal{L}$ with q_θ according to Eq. 1 {Incorporate the Imitative Model}
Initialize $\mathbf{z}_{1:T} \sim q_0$
while not converged **do**
  $\mathbf{z}_{1:T} \leftarrow \mathbf{z}_{1:T} + \nabla_{\mathbf{z}_{1:T}} \mathcal{L}(\mathbf{s}_{1:T} = f(\mathbf{z}_{1:T}), \mathcal{G}, \phi)$
end while
return $\mathbf{s}_{1:T} = f(\mathbf{z}_{1:T})$

B Routing Details

B.1 Waypointer Details

Our standard waypointer interpolates waypoints along the provided route to every 2 meters, and feeds the first 20 waypoints to the Gaussian mixture likelihood. As mentioned in the main text, one variant of our approach uses a "smart" waypointer. This waypointer simply removes nearby waypoints closer than 5 meters from the vehicle when a green light is observed in the measurements provided by CARLA, to encourage the agent to move forward, and removes far waypoints beyond 5 meters from the vehicle when a red light is observed in the measurements provided by CARLA. Note that the performance differences between our method without the smart waypointer and our method with the smart waypointer are small: the only signal in the metrics is that the smart waypointer improves the vehicle's ability to stop for red lights, however, it is quite adept at doing so without the smart waypointer.

B.2 Destination Details

For the methods we implemented, the task is to drive the *furthest* road location from the vehicle's initial position (a "Hard Destination"). Note that this protocol is somewhat more difficult than the one used in prior work [3, 16, 26, 15, 4], which has no distance guarantees between start positions and goals, and often results in much shorter paths. This difference is one of several caveats to consider in comparing the approaches directly.

C Architecture and Training Details

The architecture of $q(\mathbf{S}|\phi)$ is shown in Table 4.

Table 4: Detailed Architecture that implements $\mathbf{s}_{1:T} = f(\mathbf{z}_{1:T}, \phi)$. Typically, $T = 40$, $D = 2$,$H = W = 200$.

Component	Input [dimensionality]	Layer or Operation	Output [dimensionality]	Details
Static featurization of context: $\phi = \{\chi, \mathbf{s}^{1:A}_{-\tau:0}\}$.				
MapFeat	$\chi\ [H, W, 2]$	2D Convolution	${}^1\chi\ [H, W, 32]$	3×3 stride 1, ReLu
MapFeat	${}^{i-1}\chi\ [H, W, 32]$	2D Convolution	${}^i\chi\ [H, W, 32]$	3×3 stride 1, ReLu, $i \in [2, \ldots, 8]$
MapFeat	${}^8\chi\ [H, W, 32]$	2D Convolution	$\mathbf{\Gamma}\ [H, W, 8]$	3×3 stride 1, ReLu
PastRNN	$\mathbf{s}_{-\tau:0}\ [\tau + 1, D]$	RNN	$\alpha\ [32]$	GRU across time dimension
Dynamic generation via loop: for $t \in \{0, \ldots, T-1\}$.				
MapFeat	$\mathbf{s}_t\ [D]$	Interpolate	$\gamma_t = \mathbf{\Gamma}(\mathbf{s}_t)\ [8]$	Differentiable interpolation
JointFeat	$\gamma_t, \mathbf{s}_0, {}^2\eta, \alpha, \boldsymbol{\lambda}$	$\gamma_t \oplus \mathbf{s}_0 \oplus {}^2\eta \oplus \alpha \oplus \boldsymbol{\lambda}$	$\rho_t\ [D + 50 + 32 + 1]$	Concatenate ($\oplus$)
FutureRNN	$\rho_t\ [D + 50 + 32 + 1]$	RNN	${}^1\rho_t\ [50]$	GRU
FutureMLP	${}^1\rho_t\ [50]$	Affine (FC)	${}^2\rho_t\ [200]$	Tanh activation
FutureMLP	${}^2\rho_t [200]$	Affine (FC)	$m_t\ [D]$, $\xi_t\ [D, D]$	Identity activation
MatrixExp	$\xi_t\ [D, D]$	$\exp(\xi_t + \xi_t^{a,\text{transpose}})$	$\boldsymbol{\sigma}_t\ [D, D]$	Differentiable Matrix Exponential [21]
VerletStep	$\mathbf{s}_t, \mathbf{s}_{t-1}, m_t, \boldsymbol{\sigma}_t, \mathbf{z}_t$	$2\mathbf{s}_t - \mathbf{s}_{t-1} + m_t + \boldsymbol{\sigma}_t \mathbf{z}_t$	$\mathbf{s}_{t+1}\ [D]$	

C.1 Dataset

Before training $q(\mathbf{S}|\phi)$, we ran CARLA's expert in the dynamic world setting of Town01 to collect a dataset of examples. We have prepared the dataset of collected data for public release upon publication. We ran the autopilot in Town01 for over 900 episodes of 100 seconds each in the presence of 100 other vehicles, and recorded the trajectory of every vehicle and the autopilot's LIDAR observation. We randomized episodes to either train, validation, or test sets. We created sets of 60,701 train, 7586 validation, and 7567 test scenes, each with 2 seconds of past and 4 seconds of future position information at 10Hz. The dataset also includes 100 episodes obtained by following the same procedure in Town02.

D Baseline Details

D.1 Conditional Imitation Learning:

We designed a conditional imitation learning baselines that predicts the setpoint for the PID-controller. Each receives the same scene observations (LIDAR) and is trained with the same set of trajectories as our main method. We found the latter method very effective for stable control on straightaways. When the model encounters corners, however, prediction is more difficult, as in order to successfully avoid the curbs, the model must implicitly plan a safe path. In the latter method, we used a network architecture nearly identical to our approach's. In the static setting, we used a CILWP model that did not have access to traffic light information. In the dynamic setting, we included traffic light information, which allowed it to stop more frequently.

D.2 Model-Based Reinforcement Learning:

Static-world To compare against a purely model-based reinforcement learning algorithm, we propose a model-based reinforcement learning baseline. This baseline first learns a forwards dynamics model $\mathbf{s}_{t+1} = f(\mathbf{s}_{t-3:t}, \mathbf{a}_t)$ given observed expert data (a_t are recorded vehicle actions). We use an MLP with two hidden layers, each 100 units. Note that our forwards dynamics model does not imitate the expert preferred actions, but only models what is physically possible. Together with the same LIDAR map $\boldsymbol{\chi}$ our method uses to locate obstacles, this baseline uses its dynamics model to plan a reachability tree [12] through the free-space to the waypoint while avoiding obstacles. We plan forwards over 20 time steps using a breadth-first search search over CARLA steering angle $\{-0.3, -0.1, 0., 0.1, 0.3\}$, noting valid steering angles are normalized to $[-1, 1]$, with constant throttle at 0.5, noting the valid throttle range is $[0, 1]$. Our search expands each state node by the available actions and retains the 50 closest nodes to the waypoint. The planned trajectory efficiently reaches the waypoint, and can successfully plan around perceived obstacles to avoid getting stuck. To convert the LIDAR images into obstacle maps, we expanded all obstacles by the approximate radius of the car, 1.5 meters.

Dynamic-world We use the same setup as the Static-MBRL method, except we add a discrete temporal dimension to the search space (one $\mathbb{R}^2$ spatial dimension per T time steps into the future). All static obstacles remain static, however all LIDAR points that were known to collide with a vehicle are now removed: and replaced at every time step using a constant velocity model of that vehicle.

E Robustness Experiments details

E.1 Decoy Waypoints

In the decoy waypoints experiment, the perturbation distribution is $\mathcal{N}(0, \sigma = 8m)$: each waypoint is perturbed with a standard deviation of 8 meters. One failure mode of this approach is when decoy waypoints lie on a valid off-route path at intersections, which temporarily confuses the planner about the best route.

E.2 Plan Reliability Estimation

We determined a threshold on the planning criterion by single-goal planning to the expert's final location on offline validation data and setting it to the criterion's mean minus one stddev. Although a more intelligent calibration could be performed by analyzing the information retrieval statistics on the offline validation, we found this simple calibration to yield reasonably good performance. We used 1650 test scenes to perform classification of plans to three different types of waypoints 1) where the expert actually arrived at time T (89.4% reliable), 2) waypoints $20\mathrm{m}$ ahead along the waypointer-provided route, which are often near where the expert arrives (73.8% reliable) 3) the same waypoints from 2), shifted $2.5\mathrm{m}$ off of the road (2.5% reliable). This shows that our learned model exhibits a strong preference for valid waypoints. Therefore, a waypointer that provides expert waypoints via 1) half of the time, and slightly out-of-distribution waypoints via 3) in the other half, an "unreliable" plan classifier achieves 97.5% recall and 90.2% precision.

F Pothole Experiment Details

We simulated $2\mathrm{m}$-wide potholes in the environment by randomly inserting them in the cost map near waypoints subsampled to $20m$ with offsets distributed $\mathcal{N}(\mu{=}[-15\mathrm{m}, 2\mathrm{m}], \Sigma = \mathrm{diag}([1, 0.01]))$, (i.e. mean-centered on the right side of the lane $15\mathrm{m}$ before each waypoint).

G Baseline Visualizations

See Figure 8 for a visualization of our baseline methods.

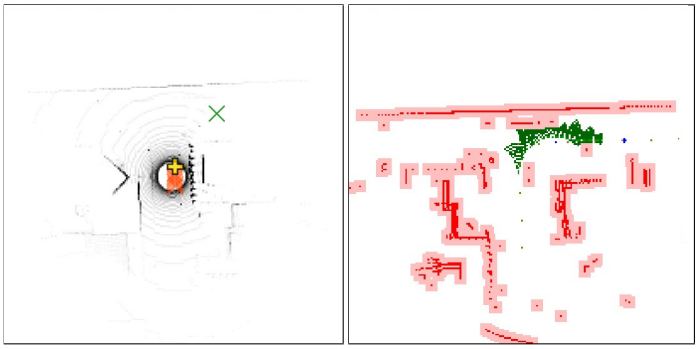

Figure 8: Baseline methods we compare against. The red crosses indicate the past 10 positions of the agent. *Left: Imitation Learning baseline*: the green cross indicates the provided goal, and the yellow plus indicates the predicted setpoint for the controller. *Right: Model-based RL baseline:* the green regions indicate the model's predicted reachability, the red regions are post-processed LIDAR used to create its obstacle map.