Learning to drive from a world on rails

Dian Chen
UT Austin

Vladlen Koltun
Intel Labs

Philipp Krähenbühl
UT Austin

Abstract

We learn an interactive vision-based driving policy from pre-recorded driving logs via a model-based approach. A forward model of the world supervises a driving policy that predicts the outcome of any potential driving trajectory. To support learning from pre-recorded logs, we assume that the world is on rails, meaning neither the agent nor its actions influence the environment. This assumption greatly simplifies the learning problem, factorizing the dynamics into a non-reactive world model and a low-dimensional and compact forward model of the ego-vehicle. Our approach computes action-values for each training trajectory using a tabular dynamic-programming evaluation of the Bellman equations; these action-values in turn supervise the final vision-based driving policy. Despite the world-on-rails assumption, the final driving policy acts well in a dynamic and reactive world. It outperforms imitation learning as well as model-based and model-free reinforcement learning on the challenging CARLA NoCrash benchmark. It is also an order of magnitude more sample-efficient than state-of-the-art model-free reinforcement learning techniques on navigational tasks in the ProcGen benchmark.

1. Introduction

Vision-based autonomous driving is hard. An agent needs to perceive, understand, and interact with its environment from incomplete and partial experiences. Most successful driving approaches [5, 27, 33, 34] reduce autonomous navigation to imitating an expert, usually a human actor. Expert actions serve as a source of strong supervision, sensory inputs of the expert trajectories explore the world, and policy learning reduces to supervised learning backed by powerful deep networks. However, expert trajectories are often heavily biased, and safety-critical observations are rare. After all, human operators drive hundreds of thousands of miles before observing a traffic incident [39]. This sparsity of safety-critical training data makes it difficult for a behavior-cloning agent to learn and recover from mistakes. Model-free reinforcement learning [25, 40] offers a solution, allowing an agent to actively explore its environment and learn from it. However, this exploration is even less data-efficient than behavior cloning, as it needs to experience mistakes to avoid them. For reinforcement learning, the required sample complexity for safe driving is prohibitively large, even in simulation [40].

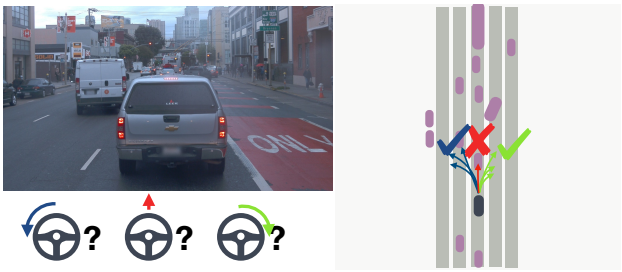
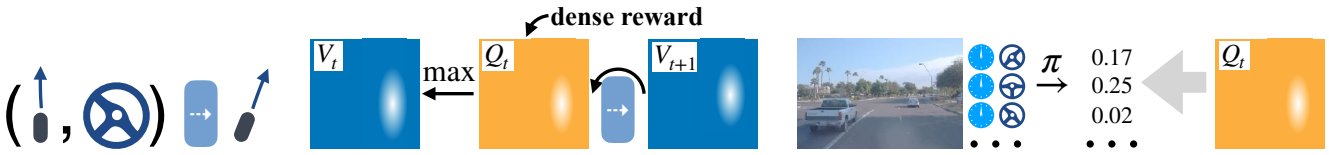


Figure 1: We learn a reactive visuomotor driving policy that gets to explore the effects of its own actions at training time. The policy simulates the effects of its own actions using a forward model in pre-recorded driving logs. It then learns to choose safe actions without explicitly experiencing unsafe driving behavior. Picture selected from the Waymo open dataset [37].

In this paper, we present a method to learn a navigation policy that recovers from mistakes without ever making them, as illustrated in Figure 1. We first learn a world-model on static pre-recorded trajectories. This world model is able to simulate the agent's actions without ever executing them. Next, we estimate action-value functions for all pre-recorded trajectories. Finally, we train a reactive visuomotor policy that gets to observe the impact of all its actions as predicted by the action-value function. The policy learns to avoid costly mistakes, or recover from them. We use driving logs, recorded lane maps and locations of traffic participants, to train the world-model and compute the action-value function. However, our visuomotor policy drives using raw sensor inputs, namely RGB images and speed readings alone. Figure 2 shows an overview.

The core challenge in our approach is to build a sufficiently expressive and accurate world-model that allows the



(a) Forward model. (b) Bellman update. (c) Distillation.

Figure 2: Overview of our approach. Given a dataset of offline driving trajectories of sensor readings, driving states, and actions, we first learn a forward model of the ego-vehicle (a). Using the offline driving trajectories, we then compute action-values under a pre-defined reward and learned forward model using dynamic programming and backward induction on the Bellman equation (b). Finally, the action-values then supervise a reactive visuomotor driving policy through policy distillation (c). For a single image, we supervise the policy for all vehicle speeds and actions for a richer supervisory signal.

agent to explore its environment and the impact of its actions.

For autonomous driving, this involves modeling the autonomous vehicles and all other traffic participants, i.e. other vehicles, pedestrians, traffic lights, etc. In its raw form, the state space in which the agent operates is too high-dimensional to effectively explore. We thus make a simplifying assumption: The agent's actions only affect its own state, and cannot directly influence the environment around it. In other words: the world is "on rails". This naturally factorizes the world-model into an agent-specific component that reacts to the agent's commands, and a passively moving world. For the agent, we learn an action-conditional forward-model. For the environment, we simply replay pre-recorded trajectories from the training data.

The factorization of the world model lends itself to a simple evaluation of the Bellman equations through dynamic programming and backward induction. For each driving trajectory, we compute a tabular approximation of the value function over all potential agent states. We use this value function and the agent's forward model to compute action-value functions, which then supervise the visuomotor policy. Action values serve as denser supervisory signals. For a single training example, we supervise the visuomotor policy on all agent states, including variations of the camera viewpoint, vehicle speed, or high level command.

We evaluate our method in the CARLA simulator [13]. On the CARLA leaderboard[1], we achieve a 25% higher driving score than the prior top-ranking entry while using $40\times$ less training data. Notably, our method uses camera-only sensors, while some prior work relies on LiDAR. We also outperform all prior methods on the NoCrash benchmark [10]. Finally, we show that our method generalizes to other environments using the ProcGen platform [7]. Our method successfully learns navigational policies in the *Maze* and *Heist* environments with an order of magnitude fewer observations than baseline algorithms. Code and data are available[2].

[1] https://leaderboard.carla.org/leaderboard/

[2] https://dotchen.github.io/world_on_rails

2. Related Work

Imitation learning is one of the earliest and most successful approaches to vision-based driving and navigation. Pomerleau [31] pioneered this direction with ALVINN. Recent work extends imitation learning to challenging urban driving and navigation in complicated environments [28, 30, 1, 9, 10, 34, 25]. Imitation learning algorithms train on trajectories collected by human experts [9, 13, 31], or privileged experts constructed with rich sensory data [5, 30]. These approaches are limited to the expert's observations and actions. In contrast, our work learns to drive from passive driving logs and integrates mental exploration into the learning process so as to imagine and learn from scenarios that were not experienced when the logs were collected.

Model-based reinforcement learning builds a forward model to help train the policy. Sutton [38], Gu et al. [15], Kalweit and Boedecker [21], Kurutach et al. [22] use a forward world model to generate imagined trajectories to improve the sample complexity. World models [29, 16, 18, 35] use the forward model to provide additional context to assist the learning agents' decision making. Feinberg et al. [14], Buckman et al. [3] roll out the forward model for short horizons to improve the fidelity of their Q or value function approximation. In our work, we factorize the forward world model into the controllable ego-agent and a passively moving environment. This factorization significantly simplifies policy learning and allows for a tabular evaluation of the Q and value functions. Our idea of factorizing the agent and the environment is similar to the idea of *exogenous events* in policy learning [2]. Recently, Dietterich et al. [12], Chitnis and Lozano-Pérez [6] considered finding a minimal factorized MDP. In contrast, we explicitly factorize the environment and focus on leveraging the factorization for planning and supervision of a visuomotor policy.

Policy distillation remaps the outputs of a privileged agent to a visuomotor agent [5, 24, 30, 23]. Levine et al. [24] use optimal control methods to learn local controllers for robotic manipulation tasks, and use them to supervise a visuomotor policy. Pan et al. [30] train a visuomotor driving

policy by imitating an MPC controller that has access to expensive sensors. Lee et al. [23] first learn a privileged policy using model-free RL, then distill a visuomotor agent. Chen et al. [5] distill a visuomotor agent from a policy learned by imitation on privileged simulator states. Our approach uses a similar privileged simulator state to infer an action-value function to supervise the final visuomotor policy. While prior work uses one policy to supervise another, in our work a tabular action-value function supervised the policy. A reactive driving policy only exists after distillation.

Cost volume based planners [41, 33, 4] score and rank select future ego-vehicle trajectories. In tabular form, they closely resemble our action-value estimate. However, our action-value estimate has two advantages. First, it supervises a policy at training time in an offline process, while cost volumes need to be predicted for inference [41, 33, 4]. Second, we make use of ground-truth states, while cost volumes use imitation [41] or affordances [33, 4] from partial observations.

3. Method

We aim to learn a reactive visuomotor policy $\pi(I)$ that produces an action $a \in \mathcal{A}$ for a sensory input I. At training time, we are given a set of trajectories $\tau \in D$. Each trajectory $\tau = \{(\hat{I}_1, \hat{L}_1, \hat{a}_1), (\hat{I}_2, \hat{L}_2, \hat{a}_2), \ldots\}$ contains a stream of sensor readings $\hat{I}_t$, corresponding driving logs $\hat{L}_t$, and executed actions $\hat{a}_t$. The hat symbols denotes data from driving logs, regular symbols denote free or random variables. The driving logs record the state (position, velocity, and orientation) of the ego-vehicle and all other traffic participants, as well as the environment state (lane information, traffic light state, etc.). We use the driving logs to compute a forward model $\mathcal{T}$ of the world and an action-value function Q from a scalar reward. The forward model $\mathcal{T}$ takes a driving state L_t and an agent's action a_t to predict the next state L_{t+1}. We use a hybrid semi-parametric model to estimate $\mathcal{T}$, as described in Section 3.1. Specifically, we factorize the forward model into a ego-vehicle component $\mathcal{T}^{ego}$ and a world component $\mathcal{T}^{world}$. We approximate the ego-vehicle forward model using a simple deep network, while the collected trajectories are used non-parametrically for the world forward model. This factorization allows us to estimate an action-value function using a tabular approximation of the Bellman equation, as described in Section 3.2. Finally, we use the estimated action-values Q to distill a visuomotor policy π. This policy π maximizes the expected return under our forward model and tabular action-value approximation. At training time, our algorithm uses privileged information, i.e. driving logs, to supervise policy learning, but the final policy $\pi(I_t)$ drives from sensor inputs alone. Algorithm 1 and Figure 2 summarize the entire training process.

Algorithm 1: Learning in a world-on-rails

Data: Training trajectories D
Result: Policy $\pi(I) \in \mathcal{A}$
`// Forward-model fitting §3.1`
Function FitForward$(D) \rightarrow \mathcal{T}^{ego}$**:**
  Minimize Equation (1);
  return ego-vehicle forward model $\mathcal{T}^{ego}$;
end
`// Action-value estimate §3.2`
Function EstimateQ$(D, \mathcal{T}^{ego}) \rightarrow Q$**:**
  for $\tau \in D$ **do**
    Initialize $V_{|\tau|+1}(\cdot) = 0$;
    for $t = |\tau| \ldots 1$ **do**
      Compute Q_t an V_t Equation (2);
      Store Q_t;
    end
  end
  return stored Q-values;
end
`// Policy distillation §3.3`
Function DistillPolicy$(D, Q) \rightarrow \pi$**:**
  Minimize Equation (3);
  return visuomotor policy π;
end
Learn forward-model $\mathcal{T}^{ego} = \textbf{FitForward}(D)$;
Estimate Action-Values $Q = \textbf{EstimateQ}(D, \mathcal{T}^{ego})$;
Learn visuomotor policy $\pi = \textbf{DistillPolicy}(D, Q)$;

3.1. A factorized forward model

In its raw form the forward model $\mathcal{T}$ is too complex to efficiently predict and simulate. After all, entire driving simulators are designed to forecast just one of the many possible future driving states. We thus factorize the driving state L_t and forward model $\mathcal{T}$ into two parts: A part considering just the vehicle being controlled $L^{ego}_{t+1} = \mathcal{T}^{ego}(L^{ego}_t, L^{world}_t, a_t)$ and a part modeling the rest of the world $L^{world}_{t+1} = \mathcal{T}^{world}(L^{ego}_t, L^{world}_t, a_t)$. Here we consider only deterministic transitions. We furthermore assume that *the world is on rails* and cannot react to the agents' commands a or the state of the ego-vehicle L^{ego}. Specifically, the transition of the world state only depends on the prior world state itself: $L^{world}_{t+1} = \mathcal{T}^{world}(L^{world}_t)$. Thus the initial state of the world L^{world}_0 determines the entire trajectory of the world: $\{L^{world}_1, L^{world}_2, \ldots\}$. This allows us to model the world transition using the collected trajectories τ directly. We thus only need to model the ego-vehicle's forward-model $\mathcal{T}^{ego}$ for any ego-vehicle state L^{ego}_t and action a_t. We train $\mathcal{T}^{ego}$ on the collected trajectories using L1

regression

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

where we roll out the forward model for $T = 10$ steps to obtain a more robust regression target. We use a simple parametric bicycle model that easily generalizes beyond the training states $\hat{L}^{ego}$, as described in Section 4.

The world-on-rails assumption clearly does not hold, neither in a simulator nor in the real world. Other agents in the world *will* react to the ego-vehicle and its actions. However, this does not imply that a world-on-rails cannot provide strong and useful supervision to the agent. Our experiments show that an agent trained in a world-on-rails significantly outperforms agents trained with a full forward model of the world. The world-on-rails assumption significantly simplifies the estimation of an action-value function in Section 3.2 and subsequent policy learning in Section 3.3.

3.2. A factorized Bellman equation

Our goal is to estimate an action-value function $Q(\hat{L}_t, a)$ for each state $\hat{L}_t$ of the training trajectory and action a. We use the Bellman equation and a tabular discretization of the value function here. Recall the γ-discounted Bellman equation: $V(L_t) = \max_a Q(L_t, a)$ and $Q(L_t, a) = \gamma V(\mathcal{T}(L_t, a)) + r(L_t, a)$ for any state L_t, action a, and reward r. Ordinarily, one would need to resort to Bellman iterations to estimate V and Q. However, our factorized forward-model simplifies this:

$$\begin{aligned} V(L^{ego}_t,\hat{L}^{world}_t) &= \max_a Q(L^{ego}_t,\hat{L}^{world}_t,a) \\ Q(L^{ego}_t,\hat{L}^{world}_t,a_t) &= r(L^{ego}_t,\hat{L}^{world}_t,a_t)+ \\ &\quad \gamma V(\mathcal{T}^{ego}(L^{ego}_t,\hat{L}^{world}_t,a),\hat{L}^{world}_{t+1}). \end{aligned}$$

The action-value function is needed for all ego-vehicle state L^{ego}, but only recorded world states $\hat{L}^{world}_t$. It is sufficient to evaluate the action-value function on just recorded world states for all ego-vehicle states: $\hat{V}_t(L^{ego}_t) = V(L^{ego}_t, \hat{L}^{world}_t)$, $\hat{Q}_t(L^{ego}_t, a_t) = V(L^{ego}_t, \hat{L}^{world}_t, a_t)$. Furthermore, the world states are strictly ordered in time, hence the Bellman equations simplifies to

$$\begin{aligned} \hat{V}_t(L^{ego}_t) &= \max_a \hat{Q}_t(L^{ego}_t,a) \quad (2) \\ \hat{Q}_t(L^{ego}_t,a_t) &= r(L^{ego}_t,\hat{L}^{world}_t,a_t)+ \\ &\quad \gamma\hat{V}_{t+1}(\mathcal{T}^{ego}(L^{ego}_t,a)). \end{aligned}$$

Here the value and action-value functions only consider recorded world states, but all possible ego-vehicle states. The model is thus able to "imagine" driving behaviors and their reward without ever executing them. In order to collect rewards from these "imagined" states, we require an explicit reward function r, and not just a scalar reward signal provided by the environment. For a detailed discussion of the reward see Section 4.

We solve Equation (2) using backward induction and dynamic programming. The state of the ego-vehicles L^{ego} is compact (position, orientation, and velocity). This allows us to compute a tabular approximation of the value function $V_t(L^{ego}_t)$, evaluated in batch operations efficiently. Specifically, we discretize $V_t(L^{ego}_t)$ into bins corresponding to the position, orientation, and velocity of the ego-vehicle. When evaluating, we use linear interpolation if the requested value falls between bins. Furthermore, the action space is also small, allowing for a discretization of the $\max$ operator in the value update. During backward induction, we implicitly represent the action-value function Q_t using V_{t+1} and the forward model $\mathcal{T}^{ego}$. We only discretize $Q_t(\hat{L}^{ego}_t, \cdot)$ to supervise the visuomotor policy at timestep t. Algorithm 1 summarizes our backward induction. More details are provided in the supplementary material for reference.

3.3. Policy Distillation

We use the action-value functions for the ego-vehicle state $Q_t(\hat{L}^{ego}_t, \cdot)$ to supervise a visuomotor policy $\pi(\hat{I}_t)$. The action-value $Q_t(\hat{L}^{ego}_t, \cdot)$ represents the expected return of an optimal policy each vehicle state. We directly optimize this expected return in our policy:

$$E_{\hat{L}^{world}_t,L^{ego}_t,\hat{I}_t}\left[\sum_a \pi(a|\hat{I}_t)\hat{Q}_t(L^{ego}_t,a)+\alpha H\Big(\pi(\cdot|\hat{I}_t)\Big)\right]. \quad (3)$$

Since the action-value functions are computed densely, only the environment needs to be recorded, not the ego state. We can therefore supervise with augmented $\hat{I}_t$ representing arbitrary L^{ego}_t. We additionally add an entropy regularizer H [17] to encourage a more diverse output policy, where α is the temperature hyperparameter. In practice, we discretize both the action-values and the visuomotor policy as described in Section 4.

4. Implementation

We implement our approach in the CARLA simulator [13] in a strictly offline manner. We first collect a static dataset by rolling out a behavior agent π_b; we use the CARLA autopilot unless specified otherwise. We use the noisy driving actions of the autopilot to learn a forward model, but do not otherwise use the autopilot as supervision.

Forward model. We train the ego-vehicle forward model $\mathcal{T}^{ego}$ on a small subset of trajectories. We collect the subset of trajectories to span the entire action space of the ego-vehicle: steering $s \in [-1, 1]$ and throttle $t \in [0, 1]$ are uniformly sampled, with brake $b \in \{0, 1\}$ sampled from a

Bernoulli distribution. The forward model $\mathcal{T}^{ego}$ takes as inputs the current ego-vehicle state as 2D location x_t, y_t, orientation θ_t, speed v_t, and predicts the next ego-vehicle state $x_{t+1}, y_{t+1}, \theta_{t+1}, v_{t+1}$. We use a parameterized bicycle model as the structural prior for $\mathcal{T}^{ego}$. In particular, we only learn the vehicle wheelbases f_b, r_b, the mapping from user steering s to wheel steering ϕ, and the mapping from throttle and braking to acceleration a. The kinematics of the bicycle model are described in the supplementary material for reference. We train $\mathcal{T}^{ego}$ in an auto-regressive manner using L1 loss and stochastic gradient descent.

Bellman equation evaluation. For each time-step t, we represent the value function V_t as a 4D tensor discretized into $N_H \times N_W$ position bins, N_v velocity bins, and N_θ orientation bins. We use $N_H = N_W = 96$, $N_v = 4$, and $N_\theta = 5$. Each bin has a physical size of $\frac{1}{3} \times \frac{1}{3} m^2$ and corresponds to a $2m/s$ velocity range and a 38° orientation range. The ego-vehicle state $\hat{L}_t^{ego} = (x_t, y_t, v, \theta)$ is always centered in this discretization. The position of the ego-vehicle (x_t, y_t) is at the center of the spatial discretization. We only represent orientations in the range $[-95^\circ, 95^\circ]$ relative to the ego-vehicle. When computing the action value function, any value V_t that does not lie in the center of a bin is interpolated among its 2^4 neighboring bins using linear interpolation. The linear interpolation is computed over all states at once and is factorized over ego state dimensions (location, speed and orientation), thus it is efficient. Values that fall outside the discretization are 0. We discretize actions into $M_s \times M_t$ bins for steering and throttle respectively, and one additional bin for braking. We do not steer or throttle while braking. We use $M_s = 9$ and $M_t = 3$ for a total of $9 \cdot 3 + 1 = 28$ discrete actions.

Policy network. The policy network uses a ResNet34 [19] backbone to parse the RGB inputs. We use global average pooling to flatten the ResNet features, before concatenating them with the ego-vehicle speed and feeding this to a fully-connected network. The network produces a categorical distribution over the discretized action space.

In CARLA, the agent receives a high-level navigation command c_t for each time-step. We supervise the visuomotor agent simultaneously on all the high-level commands [5]. Additionally, we task the agent to predict semantic segmentation as an auxiliary loss. This consistently improves the agent's driving performance, especially when generalizing to new environments.

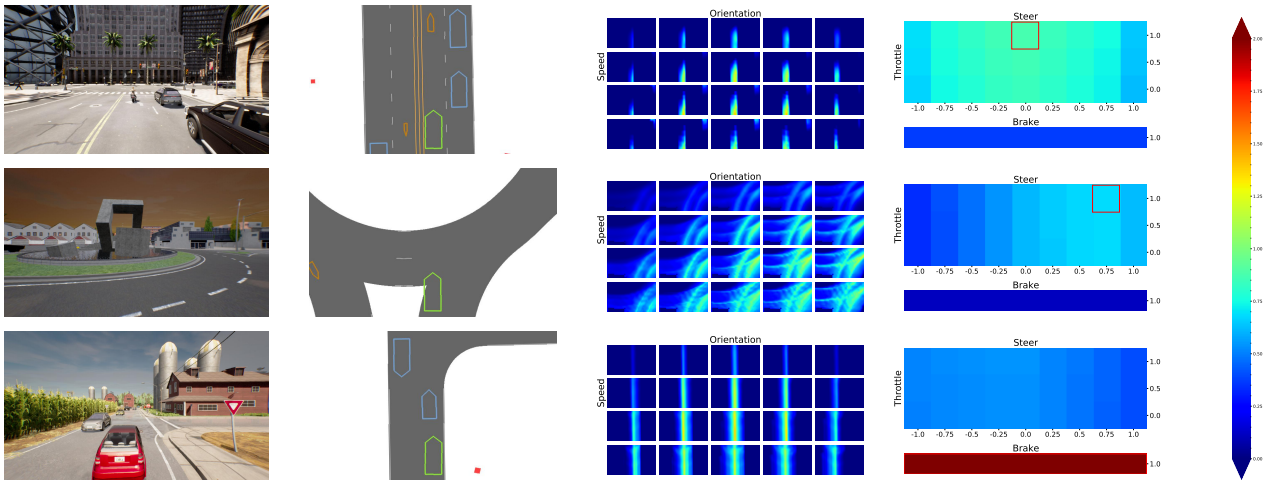
Reward design. The reward function $r(L_t^{ego}, L_t^{world}, a_t, c_t)$ considers ego-vehicle state, world state, action, and high-level command, and is computed from the driving log at each timestep. We use the lane information of the world and high-level command to first compute the target lane of the ego-vehicle. The agent receives a reward of $+1$ for staying in the target lane at the desired position, orientation and speed, and is smoothly penalized for deviating from the lane down to a value of 0. If the agent is located at a "zero-speed" region (e.g. red light, or close to other traffic participants), it is rewarded for zero velocity regardless of orientation, and penalized otherwise except for red light zones. All "zero speed" rewards are scaled by $r_{\text{stop}} = 0.01$, in order to avoid agents disregarding the target lane. The agents receives a greedy reward of $r_{\text{brake}} = +5$ if it brakes in the zero-speed zone. To avoid agents chasing braking region, the braking reward cannot be accumulated. All rewards are additive. We found that with zero-speed zones and brake rewards, there is no need to explicitly penalize collisions. We compute the action-values over all high-level commands for each timestep, and use multi-branch supervision [5] when distilling the visuomotor agent.

5. Experiments

Dataset. We evaluate our approach on the open-source CARLA simulator [13]. We train our ego-vehicle forward model on a small subset of trajectories consisting of 2400 collected frames. It learns from random actions. The bulk of our training set uses just passive sensor information I and training logs L. We refer readers to the supplement for additional details.

Experimental setup. We evaluate our approach on both the CARLA leaderboard and the NoCrash benchmark. For both benchmarks, at each frame, the agent receives RGB camera reading I, speed reading v, and a high-level command c to compute steering s, throttle t, and brake b. NoCrash benchmark consists of three driving condition; each driving condition contains 50 predefined routes: 25 for the training town (Town01) and 25 for the testing town (Town02). We refer readers to the supplement for additional details.

Comparison to the state-of-art. Table 1 compares the performance of the presented approach on the CARLA leaderboard. We list the three key metrics from the leaderboard: driving score (primary summary measure used for ranking entries on the leaderboard), route completion, and infraction score. We compare to CILRS [10], LBC [5], Transfuser [32] and IA [40]. LBC is the state of the art on the NoCrash benchmark, and Transfuser is a very recent method utilizing sensor fusion. Both LBC and Transfuser are based on imitation learning. IA is the winning entry in the 2020 CARLA Challenge, and the prior leading entry on the CARLA leaderboard. IA is based on model-free reinforcement learning with Rainbow [20] and IQN [11].



(a) RGB camera (b) Map (c) Value-map (d) Action-value

Figure 3: Visualization of the computed value function and action-value function for the current frame. The RGB camera image (a) and bird-eye's view maps (b) show the ego-vehicle location in the world. The value-maps (c) show the discretized tabular value estimate for 4 speed bins $\times$ 5 orientation bins. The orientation bins are -95° to 95° from left to right, and the speed bins are 0 m/s to 8 m/s from top to bottom. Each map has a resolution of 96×96 corresponding to a 24m^2 area around the vehicle. We crop areas behind the ego-vehicle for visualization. The value-maps use 5 bellman updates and see 1.25s into the future. (d) shows the action-values based on the current ego-vehicle state. Actions with highest values are highlighted with red boxes. These action-values supervise the visuomotor policy that takes as input camera RGB images. More visualizations provided in the supplement.

Method	DS ↑	RC ↑	IS ↑	Data	LiDAR
CILRS [10]	5.37	14.40	0.55	–	×
LBC [5]	8.94	17.54	**0.73**	–	×
Transfuser [32]	16.93	51.82	0.42	150K	✓
IA [40]	24.98	46.97	0.52	40M	×
Ours	**31.37**	**57.65**	0.56	1M	×

Table 1: Comparison of the driving score (DS, main metric), route completion (RC), and infraction score (IS) on the CARLA leaderboard. For all three metrics, higher is better. Our method improves the driving score by 25% relative to the prior state of the art [40] while using $40\times$ less data.

Table 2 compares the performance on the CARLA NoCrash benchmark. We retrain LBC (the prior state of the art on NoCrash) on CARLA 0.9.10 using the same training data with augmented camera views as in our approach. To help LBC generalize, we found it important to train with additional semantic segmentation supervision. CARLA 0.9.10 features more complex visuals, and generalization to new weather conditions is harder. IA features two models, a published model trained on CARLA 0.9.6 Town1 alone, and a much stronger CARLA Challenge model (trained on CARLA 0.9.10). We compare to the stronger challenge model. However, this model was trained on many more towns, and under both training and testing weather conditions. It thus does not have held-out testing weathers. Our method outperforms LBC and IA on all 12 tasks and conditions. Furthermore, our method does not require expert actions anywhere in the training pipeline, unlike LBC. We outperform IA on all traffic scenarios in both towns, even though we train only on Town1.

Ablation study. Table 3 compares our visuomotor agent with other model-based approaches. All baselines optimize the same reward function described in Section 4. Dreamer (DM) [18] trains a full-fledged embedding-based world model, and uses it to backpropagate analytic gradients to the policy during rollouts. Building a full forward model of our driving scenarios can be challenging. To help this baseline, we give it access to driving logs both during training and testing. We additionally construct a variant, F-DM, which utilizes our factorized world model. F-DM replaces full embedding-based world model with our ego forward model $\mathcal{T}^{ego}$. Equivalent to our method it observes the pre-recorded world states and thus cannot backpropagate through a forward model of the world. F-DM still trains the policy the same way as DM, using imaginary differentiable rollouts. Since Dreamer is off-policy, we implement both DM and F-DM in an offline RL manner, and train both on the same

Task	Town	Weather	IA	LBC	**Ours**
Empty			85	89	**98**
Regular	train	train	85	87	**100**
Dense			63	75	**96**
Empty			*77*	86	**94**
Regular	test	train	*66*	79	**89**
Dense			*33*	53	**74**
Empty			–	60	**90**
Regular	train	test	–	60	**90**
Dense			–	54	**84**
Empty			–	36	**78**
Regular	test	test	–	36	**82**
Dense			–	12	**66**

Table 2: Comparison of the success rate of the presented approach (Ours) to the state of the art on NoCrash (LBC), and the winning entry of the 2020 CARLA Challenge (IA). All three methods are trained and evaluated on CARLA 0.9.10. IA uses all towns and all weathers to train. It thus does not have test weathers. *Italic* numbers indicate that the policy was trained on the test town. Additional numbers on route completion and random seeds are provided in the supplement.

Factorized world		×	✓	✓	✓
Task	Town	DM	F-DM	CEM	**Ours**
Straight	train	*37*	*44*	**100**	**100**
Turn		*0*	*0*	*88*	**100**
Straight	test	*44*	*52*	**100**	**100**
Turn		*0*	*0*	*97*	**100**
Empty		0	0	*88*	**98**
Regular	train	*0*	*0*	*86*	**100**
Dense		*0*	*0*	*72*	**96**
Empty		*0*	*0*	**97**	94
Regular	test	*0*	*0*	*84*	**89**
Dense		*0*	*0*	*47*	**74**

Table 3: Comparison of the success rate on the CoRL17 and NoCrash benchmark under training weathers. We compare our full visuomotor agent with model-based baselines. Dreamer (DM) [18] trains the full world model, whereas the rest follow our factorization and use the same forward model $\mathcal{T}^{ego}$ as our approach. Numbers in *italic* indicate agents that use privileged information (such as driving logs) at test time. Our approach uses sensor readings alone. Nevertheless, our approach outperforms all baselines.

dataset with which we supervise our visuomotor agent. CEM is an MPC baseline that factorizes the world and uses the cross-entropy method [26] to search for the best actions. It uses our forward model, but cannot simulate the environment forward at the test time. It assumes a static world. Like Dreamer, CEM has access to the driving log at test time of the current timestep. It replans at every timestep over the most recent driving log. All the baselines use privileged information (driving logs), whereas our method takes sensor inputs alone.

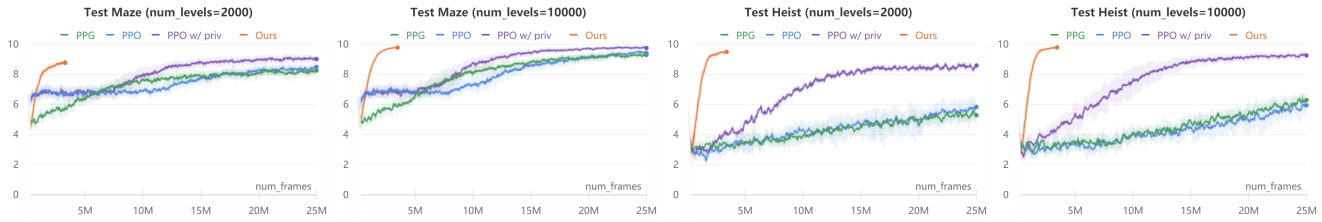
We evaluate with the training weather for our method, as driving logs for baselines are weather-agnostic[3]. We found that the NoCrash benchmark is too hard for the Dreamer baseline, and thus additionally test on the much easier CoRL17 benchmark [13]. Akin to NoCrash, each task in the CoRL17 benchmark contains 50 predefined routes: 25 for the training town and 25 for an unseen test town. It runs on empty roads with simpler routes compared to NoCrash. Our method outperforms all other model-based baselines by a margin, despite using sensor inputs instead of driving logs. Dreamer with a factorized world model outperforms the full world model but still fails to generalize beyond straight driving. One reason for the poor performance of Dreamer may be a bias in the training set. Cars mostly drive straight. Dreamer may simply see too few turning scenarios compared to the endless stream of straight driving.

Traffic light infraction analysis. We additionally analyze traffic light infractions on the NoCrash benchmark. Table 4 compares the average number of traffic light violations per hour on all trials in the NoCrash benchmark. Our method has fewer traffic light infractions than the reinforcement learning baseline (IA) on all six tasks under the training weathers.

Visualization. Figure 3 shows a visualization of the computed value and action-value functions for various driving scenarios. Each of these action-value functions densely supervises policy for the displayed image.

ProcGen navigation. To demonstrate the broad applicability of our approach, we additionally evaluate on the navigational tasks (*Maze* and *Heist*) in the ProcGen benchmark [7]. In both environments, the agent is rewarded for navigating to the desired locations. *Maze* features a plain navigation task through a complex environment. *Heist* additionally requires the agent to collect keys and unlock the doors before navigating to the goal. In ProcGen, the action space is discrete, hence we only discretize the ego-agent's states. We ignore velocity. The agent's forward dynamics model in ProcGen is not location agnostic as in CARLA. To address this, we use a small ConvNet to extract the environment context around

[3]The physics in CARLA does not vary with weather. Only sensor readings change with different weather conditions



(a) Maze 2000 training levels (b) Maze 10000 training levels (c) Heist 2000 training levels (d) Heist 10000 training levels

Figure 4: Comparison of our method to state-of-the-art model-free reinforcement learning on the navigational tasks of the ProcGen benchmark. All plots measure the average episode returns on the testing levels. PPO w/ priv is a customized PPO implementation that during training additionally takes as input the same privileged information that our approach uses to compute rewards and train the agent forward model. The presented approach is an order of magnitude more sample-efficient.

the ego-agent forward model $\mathcal{T}^{ego}$. In order to evaluate sample efficiency, we implement our method on ProcGen in an off-policy reinforcement learning manner. We alternate between training or fine-tuning a policy and forward model, and rolling out new trajectories under the current policy. Compared to model-free baselines, our approach needs access to a dense reward function, instead of just the scalar reward signal of the environment. We compute this reward function using semantic labels obtained via the ProcGen renderer. For *Maze*, the reward function awards $+1$ for goal location regardless of orientation. For *Heist*, the reward function awards $+1$ for key and unlockable door locations regardless of orientation. In addition, we mask all unachievable ego-state values to 0 during the Bellman equation evaluation. We use this privileged information in the action-value computation only, and in no other place in our algorithm. Figure 4 compares the performance and sample-efficiency of our method with model-free reinforcement learning baselines PPO [36] and PPG [8]. PPG is the current state of the art on the ProcGen benchmark. In addition, we compare to a customized PPO implementation which during training also takes as input the same privileged information used in our method. Our method converges within 3M frames, while model-free baselines take up to 25M frames. For both *Maze* and *Heist* environments, we train all agents on two different conditions: 2000 and 10000 (procedurally generated) training levels. For both environments, agents are tested on completely randomized procedurally-generated levels. The comparison of average episode returns on the training levels is in the supplement for reference. Our method is an order of magnitude more sample-efficient than all the model-free RL baselines even when those methods are given the same privileged information used by our reward computation.

Oracle actions			×	✓	×
Task	Town	Weather	IA	LBC	**Ours**
Empty			3.34	1.35	**0.00**
Regular	train	train	6.71	1.89	**0.43**
Dense			15.41	3.27	**2.61**
Empty			62.18	**8.45**	10.68
Regular	test	train	53.28	8.22	**6.95**
Dense			54.94	**7.26**	12.90
Empty			–	0.36	**0.00**
Regular	train	test	–	0.81	**0.00**
Dense			–	**0.52**	4.29
Empty			–	**8.17**	14.46
Regular	test	test	–	**8.61**	11.30
Dense			–	**4.87**	13.28

Table 4: Comparison of the average number of traffic light violations per hour of trials on the NoCrash benchmark. We compare our approach to LBC (prior state of the art on NoCrash) and IA (the winning entry of the 2020 CARLA challenge). LBC trains from oracle trajectories, whereas IA and ours do not.

6. Conclusion

We show that assuming independence between the agent and the environment, which we refer to as a *world on rails*, significantly simplifies modern reinforcement learning. While true independence rarely holds, the gains in training efficacy outweigh the modeling constraints. Even with a simple reward function, an agent trained in a world-on-rails learns to drive better than state-of-the-art imitation learning agents on standard benchmarks. In addition, the presented policy learning framework is an order of magnitude more sample-efficient than state-of-the-art reinforcement learning on challenging ProcGen navigation tasks.

Acknowledgements

We thank Yuke Zhu for his valuable feedback. We thank Tianwei Yin for his help on figure 1. This work was supported by the NSF Institute for Foundations of Machine Learning and NSF award #1845485.

References

[1] Mayank Bansal, Alex Krizhevsky, and Abhijit Ogale. Chauffeurnet: Learning to drive by imitating the best and synthesizing the worst. In *RSS*, 2019. 2

[2] Craig Boutilier, Thomas Dean, and Steve Hanks. Decision-theoretic planning: Structural assumptions and computational leverage. In *JAIR*, 1999. 2

[3] Jacob Buckman, Danijar Hafner, George Tucker, Eugene Brevdo, and Honglak Lee. Sample-efficient reinforcement learning with stochastic ensemble value expansion. In *NeurIPS*, 2018. 2

[4] Sergio Casas, Abbas Sadat, and Raquel Urtasun. Mp3: A unified model to map, perceive, predict and plan. *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021. 3

[5] Dian Chen, Brady Zhou, Vladlen Koltun, and Philipp Krähenbühl. Learning by cheating. In *CoRL*, 2019. 1, 2, 3, 5, 6

[6] Rohan Chitnis and Tomás Lozano-Pérez. Learning compact models for planning with exogenous processes. In *CoRL*, 2020. 2

[7] Karl Cobbe, Christopher Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. In *arXiv preprint*, 2019. 2, 7

[8] Karl Cobbe, Jacob Hilton, Oleg Klimov, and John Schulman. Phasic policy gradient. In *arXiv preprint*, 2020. 8

[9] Felipe Codevilla, Matthias Müller, Antonio López, Vladlen Koltun, and Alexey Dosovitskiy. End-to-end driving via conditional imitation learning. In *ICRA*, 2018. 2

[10] Felipe Codevilla, Eder Santana, Antonio M López, and Adrien Gaidon. Exploring the limitations of behavior cloning for autonomous driving. In *ICCV*, 2019. 2, 5, 6

[11] Will Dabney, Georg Ostrovski, David Silver, and Rémi Munos. Implicit quantile networks for distributional reinforcement learning. In *ICML*, 2018. 5

[12] Thomas Dietterich, George Trimponias, and Zhitang Chen. Discovering and removing exogenous state variables and rewards for reinforcement learning. In *ICML*, 2018. 2

[13] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *CoRL*, 2017. 2, 4, 5, 7

[14] Vladimir Feinberg, Alvin Wan, Ion Stoica, Michael I Jordan, Joseph E Gonzalez, and Sergey Levine. Model-based value estimation for efficient model-free reinforcement learning. In *arXiv preprint*, 2018. 2

[15] Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine. Continuous deep q-learning with model-based acceleration. In *ICML*, 2016. 2

[16] David Ha and Jürgen Schmidhuber. Recurrent world models facilitate policy evolution. In *NeurIPS*, 2018. 2

[17] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *ICML*, 2018. 4

[18] Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. In *ICLR*, 2019. 2, 6, 7

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 5

[20] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *AAAI*, 2018. 5

[21] Gabriel Kalweit and Joschka Boedecker. Uncertainty-driven imagination for continuous deep reinforcement learning. In *CoRL*, 2017. 2

[22] Thanard Kurutach, Ignasi Clavera, Yan Duan, Aviv Tamar, and Pieter Abbeel. Model-ensemble trust-region policy optimization. In *ICLR*, 2018. 2

[23] Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. Learning quadrupedal locomotion over challenging terrain. In *Science robotics*, 2020. 2, 3

[24] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. In *JMLR*, 2016. 2

[25] Xiaodan Liang, Tairui Wang, Luona Yang, and Eric Xing. Cirl: Controllable imitative reinforcement learning for vision-based self-driving. In *ECCV*, 2018. 1, 2

[26] Shie Mannor, Reuven Y Rubinstein, and Yohai Gat. The cross entropy method for fast policy search. In *ICML*, 2003. 7

[27] Matthias Mueller, Alexey Dosovitskiy, Bernard Ghanem, and Vladlen Koltun. Driving policy transfer via modularity and abstraction. In *CoRL*, 2018. 1

[28] Urs Muller, Jan Ben, Eric Cosatto, Beat Flepp, and Yann L Cun. Off-road obstacle avoidance through end-to-end learning. In *NeurIPS*, 2006. 2

[29] Junhyuk Oh, Satinder Singh, and Honglak Lee. Value prediction network. In *NeurIPS*, 2017. 2

[30] Yunpeng Pan, Ching-An Cheng, Kamil Saigol, Keuntak Lee, Xinyan Yan, Evangelos Theodorou, and Byron Boots. Agile autonomous driving using end-to-end deep imitation learning. In *RSS*, 2018. 2

[31] Dean A Pomerleau. Alvinn: An autonomous land vehicle in a neural network. In *NeurIPS*, 1989. 2

[32] Aditya Prakash, Kashyap Chitta, and Andreas Geiger. Multi-modal fusion transformer for end-to-end autonomous driving. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021. 5, 6

[33] Abbas Sadat, Sergio Casas, Mengye Ren, Xinyu Wu, Pranaab Dhawan, and Raquel Urtasun. Perceive, predict, and plan: Safe motion planning through interpretable semantic representations. In *ECCV*, 2020. 1, 3

[34] A Sauer, N Savinov, and A Geiger. Conditional affordance learning for driving in urban environments. In *CoRL*, 2018. 1, 2

[35] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. In *Nature*, 2020. 2

[36] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. In *arXiv preprint*, 2017. 8

[37] Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurelien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin Caine, et al. Scalability in perception for autonomous driving: Waymo open dataset. In *CVPR*,

2020. 1

[38] Richard S Sutton. Planning by incremental dynamic programming. In *PMLR*, 1991. 2

[39] Brian Tefft. Rates of motor vehicle crashes, injuries and deaths in relation to driver age, united states, 2014-2015. In *AAA Foundation for Traffic Safety.*, 2017. 1

[40] Marin Toromanoff, Emilie Wirbel, and Fabien Moutarde. End-to-end model-free reinforcement learning for urban driving using implicit affordances. In *CVPR*, 2020. 1, 5, 6

[41] Wenyuan Zeng, Wenjie Luo, Simon Suo, Abbas Sadat, Bin Yang, Sergio Casas, and Raquel Urtasun. End-to-end interpretable neural motion planner. In *CVPR*, 2019. 3