ROBUST IMITATION VIA DECISION-TIME PLANNING

Anonymous authors
Paper under double-blind review

ABSTRACT

The goal of imitation learning is to mimic expert behavior from demonstrations, without access to an explicit reward signal. A popular class of approach infers the (unknown) reward function via inverse reinforcement learning (IRL) followed by maximizing this reward function via reinforcement learning (RL). The policies learned via these approaches are however very brittle in practice and deteriorate quickly even with small test-time perturbations due to compounding errors. We propose *Imitation with Planning at Test-time* (IMPLANT), a new algorithm for imitation learning that utilizes decision-time planning to correct for compounding errors of any base imitation policy. In contrast to existing approaches, we retain both the imitation policy and the rewards model at decision-time, thereby benefiting from the learning signal of the two components. Empirically, we demonstrate that IMPLANT significantly outperforms benchmark imitation learning approaches on standard control environments and excels at zero-shot generalization when subject to challenging perturbations in test-time dynamics.

1 INTRODUCTION

The objective of imitation learning is to optimize agent policies directly from demonstrations of expert behavior. Such a learning paradigm sidesteps reward engineering, which is a key bottleneck for applying reinforcement learning (RL) in many real-world domains, *e.g.*, autonomous driving, robotics. In the presence of a finite dataset of expert demonstrations however, a key challenge with current approaches is that the learned policies can quickly deviate from intended expert behavior and lead to compounding errors at test-time (Osa et al., 2018). Moreover, it has been observed that imitation policies can be brittle and drastically deteriorate in performance with even small perturbations to the dynamics during execution (Christiano et al., 2016; de Haan et al., 2019).

A predominant class of approaches to imitation learning is based on inverse reinforcement learning (IRL) and involve successive application of two steps: (a) an IRL step where the agent infers the (unknown) reward function for the expert, followed by (b) an RL step where the agent maximizes the inferred reward function via a policy optimization algorithm. For example, many popular IRL approaches consider an adversarial learning framework (Goodfellow et al., 2014), where the reward function is inferred by a discriminator that distinguishes expert demonstrations from roll-outs of an imitation policy [IRL step] and the imitation agent maximizes the inferred reward function to best match the expert policy [RL step] (Ho & Ermon, 2016; Fu et al., 2017). In this sense, reward inference is only an intermediary step towards learning the expert policy and is discarded post-training of the imitation agent.

We introduce *Imitation with Planning at Test-time* (IMPLANT), a new algorithm for imitation learning that incorporates decision-time planning within an IRL algorithm. During training, we can use any standard IRL approach to estimate a reward function and a stochastic imitation policy, along with an additional value function. The value function can be learned explicitly or is often a byproduct of standard RL algorithms that involve policy evaluation, such as actor-critic methods (Konda & Tsitsiklis, 2000; Peters & Schaal, 2008). At decision-time, we use the learned imitation policy in conjunction with a closed-loop planner. For any given state, the imitation policy proposes a set of candidate actions and the planner estimates the returns for each of actions by performing fixed-horizon rollouts. The rollout returns are estimated using the learned reward and value functions. Finally, the agent picks the action with the highest estimated return and the process is repeated at each of the subsequent timesteps.

Conceptually, IMPLANT aims to counteract the imperfections due to policy optimization in the RL step by using the reward function (along with a value function) estimated in the IRL step for decision-time planning. We demonstrate strong empirical improvements using this approach over benchmark imitation learning algorithms in a variety settings derived from the MuJoCo-based benchmarks in OpenAI Gym (Todorov et al., 2012; Brockman et al., 2016). In default evaluation setup where train and test environments match, we observe that IMPLANT improves by 16.5% on average over the closest baseline.

We also consider transfer setups where the imitation agent is deployed in test dynamics that differ from train dynamics and the test dynamics are inaccessible to the agent during both training and decision-time planning. In particular, we consider the following three setups: (a) "causal confusion" where the agent observes nuisance variables in the state representation during training (de Haan et al., 2019), (b) motor noise which adds noise in the executed actions during testing (Christiano et al., 2016), and (c) transition noise which adds noise to the next state distribution during testing. In all these setups, we observe that IMPLANT consistently and robustly transfers to test environments with improvements of 35.2% on average over the closest baseline.

2 PRELIMINARIES

Problem Setup. We consider the framework of Markov Decision Processes (MDP) (Puterman, 1990). An MDP is denoted by a tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, p_0, r, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}_{\geq 0}$ are the stochastic transition dynamics, $p_0 : \mathcal{S} \to \mathbb{R}_{\geq 0}$ is the initial state distribution, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, and $\gamma \in [0, 1)$ is the discount factor. We assume an infinite horizon setting. At any given state $s \in \mathcal{S}$, an agent makes decisions via a stochastic policy $\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}_{\geq 0}$. We denote a trajectory to be a sequence of state-action pairs $\tau = (s_0, a_0, s_1, a_1, \cdots)$. Any policy π, along with MDP parameters, induces a distribution over trajectories, which can be expressed as $p_\pi(\tau) = p(s_0) \prod_{t=0}^{\infty} \pi(a_t|s_t)\mathcal{T}(s_{t+1}|s_t, a_t)$. The return of a trajectory is the discounted sum of rewards $R(\tau) = \sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)$.

In reinforcement learning (RL), the goal is to learn a parameterized policy π_θ that maximizes the expected returns w.r.t. the trajectory distribution. Maximizing such an objective requires interaction with the underlying MDP for simulating trajectories and querying rewards. However, in many high-stakes scenarios, the reward function is not directly accessible and hard to manually design.

In imitation learning, we sidestep the availability of the reward function. Instead, we have access to a finite set of D trajectories τ_E (a.k.a. demonstrations) that are sampled from an expert policy π_E. Every trajectory $\tau \in \tau_E$ consists of a finite length sequence of state and action pairs $\tau = (s_0, a_0, s_1, a_1, \cdots)$, where $s_0 \sim p_0(s)$, $a_t \sim \pi_E(\cdot|s_t)$, and $s_{t+1} \sim \mathcal{T}(\cdot|s_t, a_t)$. Our goal is to learn a parameterized policy π_θ which best approximates the expert policy given access to τ_E. Next, we discuss the two major families of techniques for imitation learning.

2.1 BEHAVIORAL CLONING

Behavioral cloning (BC) casts imitation learning as a supervised learning problem over state-action pairs provided in the expert demonstrations (Pomerleau, 1991). In particular, we learn the policy parameters by solving a regression problem with states s_t and actions a_t as the features and target labels respectively. Formally, we minimize the following objective:

$$\ell_{BC}(\theta) := \sum_{(s_t, a_t) \in \tau_E} \|a_t - \pi_\theta(s_t)\|_2^2. \tag{1}$$

In practice, BC agents suffer from *distribution shift* in high dimensions, where small deviations in the learned policy quickly accumulate during deployment and lead to a significantly different trajectory distribution relative to the expert (Ross & Bagnell, 2010; Ross et al., 2011).

2.2 INVERSE REINFORCEMENT LEARNING

An alternative indirect approach to imitation learning is based on inverse reinforcement learning (IRL). Here, the goal is to infer a reward function for the expert and subsequently maximize the inferred reward to obtain a policy. For brevity, we focus on adversarial imitation learning approaches

to IRL (Goodfellow et al., 2014). These approaches represent the state-of-the-art in imitation learning and are also relevant baselines for our empirical evaluations.

Generative Adversarial Imitation Learning (GAIL) is an IRL algorithm that formulates imitation learning as an "occupancy measure matching" objective w.r.t. a suitable probabilistic divergence (Ho & Ermon, 2016). GAIL consists of two parameterized networks: (a) a policy network π_θ (generator) which is used to rollout agent trajectories (assuming access to transition dynamics), and (b) a discriminator D_ϕ which distinguishes between "real" expert demonstrations and "fake" agent trajectories. Given expert trajectories τ_E and agent trajectories τ_θ, the discriminator minimizes the cross-entropy loss:

$$\ell_{IRL}(\phi) := -\mathbb{E}_{\tau_E}\left[\log D_\phi(\tau_E)\right] - \mathbb{E}_{\tau_\theta}\left[\log(1 - D_\phi(\tau_\theta))\right]. \tag{2}$$

We then feed the discriminator output $-\log(1-D_\phi(s,a))$ as the inferred reward function to the generator policy. The policy parameters θ can be updated via any regular policy optimization algorithm for the RL objective, *e.g.*, Ho & Ermon (2016) use the TRPO algorithm (Schulman et al., 2015). By simulating agent rollouts, GAIL seeks to match the full trajectory state-action distribution of the imitation agent with the expert as opposed to BC which greedily matches the conditional distribution of individual actions given the states. In practice, GAIL and its variants (Li et al., 2017; Fu et al., 2017) outperform BC but might need excessive interactions with the training environment for sampling rollouts during training. Crucially, both BC and IRL approaches tend to fail catastrophically in the presence of small perturbations and nuisances at test-time (de Haan et al., 2019).

3 THE IMPLANT FRAMEWORK

In the previous section, we showed that current IRL algorithms consider reward inference as an auxiliary task for imitation learning. Once the agents have been trained, the reward function is discarded and the learned policy is deployed.[1] Indeed, if the RL step post reward inference (*e.g.*, generator updates in GAIL) were optimal, then the reward function provides no additional information about the expert relative to the imitation policy. However, this is far from reality, as current RL algorithms can fail to return optimal solutions due to either representational or optimization issues. For example, there might be a mismatch in the architecture of the policy network and the expert policy, and/or difficulties in optimizing non-convex objective functions. In fact, the latter challenge gets exacerbated in adversarial learning scenarios due to a non-stationary reward.

Building off these observations, we propose *Imitation with Planning at Test-time* (IMPLANT), an imitation learning algorithm that employs the learned reward function for decision-time planning. The pseudocode for IMPLANT is shown in Algorithm 1. We can dissect IMPLANT into two sequential phases: a training phase and a planning phase.

Training phase: We can invoke any IRL algorithm, *e.g.*, GAIL to optimize for a stochastic imitation policy π_θ by optimizing for some inferred reward function r_ϕ. Additionally, we also train a parameterized value function V_ψ at this stage. Value function estimation is often a subroutine for many RL algorithms including those which are used to update the policy within the IRL setup, such as actor-critic methods (Konda & Tsitsiklis, 2000). For such algorithms, learning a value function does not incur any additional computation.

Planning phase: At decision-time, we use the imitation policy along with the learned value and reward functions for closed-loop planning. We build our planner based on model-predictive control (MPC) (Camacho & Alba, 2013). At any given state s_t and time $t \geq 0$, we are interested in choosing action sequences for trajectories which maximizes the following objective:

$$a_t, a_{t+1}, \cdots, = \underset{a_t, a_{t+1}, \cdots}{\arg\max}\, R(\tau) = \sum_{t'=t}^{\infty} \gamma^{t'-t} r(s_{t'}, a_{t'}) \tag{3}$$

where $s_0 \sim p_0$ and $s_{t+1} \sim \mathcal{T}(\cdot|s_t, a_t)$ for all $t \geq 0$.

[1] In some cases, the reward function is transferred to a new environment and a new policy is learned using the reward function and *additional interactions* with the new environment. See Section 5 for further discussion.

Algorithm 1: *Imitation with Planning at Test-time* (IMPLANT)

Input: available dynamics $\hat{\mathcal{T}}$, expert demonstrations τ_E, rollout budget B, rollout policy π, horizon H, test start state s_0
Note: For brevity, we omit relevant MDP parameters in the list of arguments
Function `Train(`τ_E`)`**:**
    Learn a policy π_θ and a reward function r_ϕ with any existing IRL algorithm given access to demonstrations τ_E , *e.g.*, GAIL
    Estimate a value function V_ψ for π_θ
    return π_θ, r_ϕ, V_ψ;
Function `Plan(`s, π_θ, V_ψ, r_ϕ, π, H, B, $\hat{\mathcal{T}}$`)`**:**
    Set $s = s_0$
    while *agent is alive* **do**
        // Agent planning
        Sample B trajectories $\{\tau^{(1)}, \tau^{(2)}, ..., \tau^{(B)}\}$ of max length H starting from s using dynamics $\hat{\mathcal{T}}$; sample the first action $a_0^{(i)} \sim \pi_\theta$, and sample subsequent actions from π as $a_{>0}^{(i)} \sim \pi$, for $i \in \{1, 2, ..., B\}$
        Estimate trajectory returns $\hat{R}_{\phi,\psi}(\tau^{(i)})$ using V_ψ and r_ϕ (see Eq. 4)
        Pick best action index $i^* = \arg\max_i \hat{R}_{\phi,\psi}(\tau^{(i)})$ and execute the best action $a_0^{(i^*)}$
        // Environment feedback
        Observe true reward $r(s, a_0^{(i^*)})$ and true next state $s \sim \mathcal{T}(\cdot|s, a_0^{(i^*)})$
    end

This objective has also been applied for model-based RL with a learned dynamics model and black-box access to the rewards function (Nagabandi et al., 2018; Chua et al., 2018). Unlike the RL setting however, we do not know the reward function for imitation learning. The true dynamics model may be available for planning (*i.e.*, $\hat{\mathcal{T}} = \mathcal{T}$) as in Ho & Ermon (2016) or can be estimated from expert demonstrations or online interactions (Baram et al., 2016). Hence, we can do rollouts as before in regular model-based RL but need to rely on learned estimates for the reward function. In particular, we use the learned reward function r_ϕ up to a fixed horizon H and a terminal value function V_ψ thereafter to estimate the trajectory return as:

$$R(\tau) \approx \sum_{t'=t}^{t+H-1} \gamma^{t'-t} r_\phi(s_{t'}, a_{t'}) + \gamma^H V_\psi(s_H) := \hat{R}_{\phi,\psi}(\tau). \tag{4}$$

Substituting Eq. 4 in Eq. 3, we obtain a surrogate objective for optimization. To optimize this surrogate, we propose a variant of the random shooting optimizer (Richards, 2005) that works as follows. At the current state s_t, we first sample a set of B candidate actions independently from the imitation policy. For each candidate action, we estimate a score based on their expected returns by performing rollout(s) of fixed-length H. The rollout policy π from which we sample all subsequent actions could be random (potentially high variance) or the imitation policy π_θ (potentially high bias) or a mixture. In our experiments, we obtained consistently better performance with using π_θ as the rollout policy π. For each trajectory, we estimate its return via Eq. 4 and finally, pick the action with the largest return.

Consistent with the closed-loop nature of MPC, we repeat the above procedure at the next state s_t. Doing so helps correct for errors in estimation and optimization in the previous time step, albeit at the expense of additional computation. The algorithm has two critical parameters that induce similar computational trade-offs. First, we need to specify a budget B for the total number of rollouts. The higher the budget, larger is our search space for the best action. Second, we need to specify a planning horizon H. For larger lengths, we need extra computation that also involves interactions with the dynamics of the environment and rely more on the learned reward function than the value function for estimating returns in Eq. 4. However, since the rollouts are independent, we can mitigate additional computational costs by parallelizing the rollouts. While this parallelization

Table 1: Average return of imitation learning algorithms on MuJoCo benchmarks.

	Hopper	HalfCheetah	Walker2d
Expert	3570	891	3593
BC	127 ± 85	427 ± 131	258 ± 262
BC-Dropout	169 ± 105	542 ± 275	1622 ± 861
GAIL	3506 ± 337	954 ± 282	2780 ± 1007
GAIL-Reward Only	319 ± 123	5 ± 117	56 ± 146
IMPLANT (ours)	**3633** ± 50	**1193** ± 143	**3360** ± 442

is indeed bottlenecked by the rollout with the largest horizon, in all of our experiments, we perform rollouts of fixed length and the horizon that corresponds to the optimal performance is relatively small $(10 \sim 50)$. Thus, the gains due to parallelization are significant.

In the next section, we present our empirical validation that also investigates the effect of planning horizon on the performance of the algorithm in greater detail.

4 EXPERIMENTS

Our experiments aim to evaluate the performance of IMPLANT as a standalone imitation algorithm in two kinds of settings. First, we evaluate its performance in the default "no-transfer" setting, where the agent is trained and tested in the same environment. Second, we emphasize the robustness of IMPLANT by evaluating its zero-shot generalization performance in environments where the test dynamics are a perturbed version of the training dynamics. We consider 3 such perturbations: causal confusion (de Haan et al., 2019), motor noise (Christiano et al., 2016), and transition noise. We will describe each of these setups subsequently alongside the results. For all transfer settings, we only assume access to the training dynamics $\mathcal{T}_{train}$ and use it as $\hat{\mathcal{T}}$ for planning. At test-time, no additional interactions is allowed, nor do we have access to the test dynamics $\mathcal{T}_{test}$.

Setup. We evaluate our approach on MuJoCo enviroments in OpenAI Gym (Brockman et al., 2016): Hopper, HalfCheetah, and Walker2d. The expert data used for benchmarking imitation learning on these environments is publicly available[2]. We replicate the experimental setup of Ho & Ermon (2016) by fixing a limited number of expert trajectories used for training, as well as sub-sampling expert trajectories every 20 time steps. All results are averaged over 5 runs of each algorithm with different seeds. We provide further details in Appendix A.

Baselines. As we observed in Algorithm 1, IMPLANT can employ any IRL algorithm under the hood. For our experiments, we consider GAIL (Ho & Ermon, 2016) as the IRL algorithm of choice both as input for IMPLANT and consequently, as the closest baseline of interest. GAIL is amongst the current state-of-the-art methods for imitation learning; see Section 2.2 for a detailed description. For every environment, we report results for IMPLANT using a single set of hyperparameters for the rollout budget and planning horizon. We provide further details in Appendix A.

In addition, we also consider a Behavioral Cloning (BC) baseline; see Section 2.1 for a detailed description. Further, we also tested two variants of GAIL and BC that employ dropout (Srivastava et al., 2014) to demonstrate the limited utility of standard regularization techniques in countering the challenges due to low data and test noise. In fact, GAIL with dropout completely failed to learn in the adversarial setting on any of the environments; for brevity, we exclude it from presentation.

Last, we include a "GAIL-Reward Only" ablation baseline where we discard the imitation policy (generator) of GAIL during execution and instead, only use the inferred reward model (discriminator) in conjunction with a random policy for decision-time planning. This directly contrasts with the GAIL baseline, which by default only uses the generator. On the other hand, IMPLANT uses both the generator and discriminator for imitation via decision-time planning.

[2] `https://github.com/openai/baselines`

Table 2: Average return of imitation learning algorithms in causal confusion setting.

	Hopper	HalfCheetah	Walker2d
Expert	3570	891	3593
BC	209 ± 121	331 ± 141	119 ± 206
BC-Dropout	162 ± 108	548 ± 175	700 ± 433
GAIL	579 ± 484	699 ± 200	613 ± 465
GAIL-Reward Only	515 ± 302	-82 ± 79	57 ± 146
IMPLANT (ours)	**1717** ± 1262	**827** ± 375	**807** ± 395

4.1 Imitation with Limited Expert Trajectories

With a relatively low number of expert trajectories, it has been shown by Ho & Ermon (2016) that GAIL can achieve near-expert performance in almost all these environments. We evaluate the performance of IMPLANT using the lowest number of expert trajectories tested in prior work. The results are shown in Table 1. We find that IMPLANT can achieve near-optimal performance on all these environments, including Walker2d where GAIL performs much worse than the expert. As expected, BC and BC and BC-Dropout perform poorly in this setting. GAIL-Reward Only exhibits the poorest performance suggesting the benefits of explicitly learning a parametric policy.

4.2 Zero-Shot Transfer Setting: Causal confusion

de Haan et al. (2019) observed that imitation learning approaches are susceptible to *causal confusion*, *i.e.*, their performance deteriorates significantly in the presence of nuisance confounders in the state representation. To demonstrate this phenomena empirically, de Haan et al. (2019) further propose a challenging example setup in which the nuisance can be created by appending the agent's observation with its action from the previous time step. A standard imitation agent trained in this environment will learn to *copy* the previous action (since successive actions are highly correlated in expert demonstrations), falling prey to causal confusion. At test-time, the agent's performance drops drastically if the appended action is replaced by random noise (*i.e.*, the confounding is removed). We refer the reader to de Haan et al. (2019) for further details and analysis.

We now benchmark the zero-shot test performance of IMPLANT under the same setup in Table 2. While all baselines, including GAIL, fail drastically due to the confounding nuisance, IMPLANT is significantly more robust in all environments. We can visualize the agent performance qualitatively in Figure 3. Note that we provided the IMPLANT agent access to only the confounded dynamics for decision-time planning. The algorithm is hence zero-shot, unlike the proposed solutions of Fu et al. (2017) and de Haan et al. (2019) which require further interactions with the non-confounded test environment for recovery.

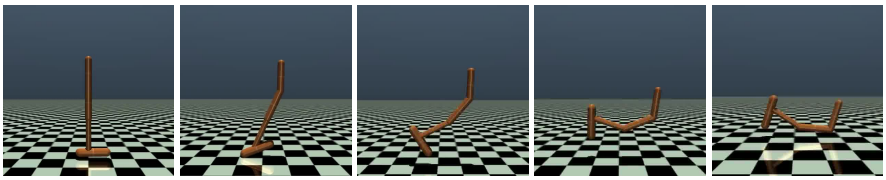
4.3 Zero-Shot Transfer Setting: Observation and action noise

Next, we consider two kinds of noisy perturbations motivated by real-world applications in sim2real.

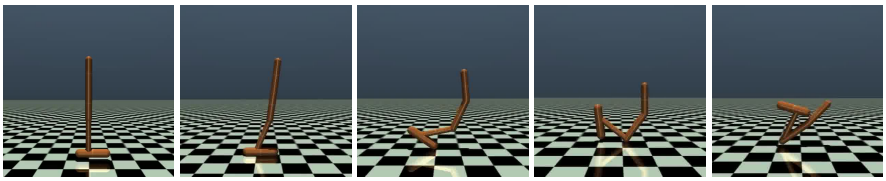
First, we perturb the intended actions via *motor noise* (Christiano et al., 2016), *e.g.*, due to imperfect hardware, a real robot might execute a noisy version of the action proposed by the agent. We implement this scenario by adding independent Gaussian noise to each dimension of the executed action at test-time, *i.e.*, $\epsilon_{\text{action}} \overset{i.i.d.}{\sim} \mathcal{N}(0, \sigma^2)$ and we vary the noise stddev $\sigma \in [0.1, 0.2, 0.5, 1.0]$.

Second, we consider *transition noise* due to an imperfect dynamics model for a simulator that may not be able to account for perturbations due to drag or friction. Hence, we specify the test-time dynamics to be a perturbed noisy version of the training dynamics. Similar to motor noise, we sample the transition noise as $\epsilon_{\text{transition}} \overset{i.i.d.}{\sim} \mathcal{N}(0, \sigma^2)$ with $\sigma \in [0.001, 0.002, 0.005, 0.01]$.

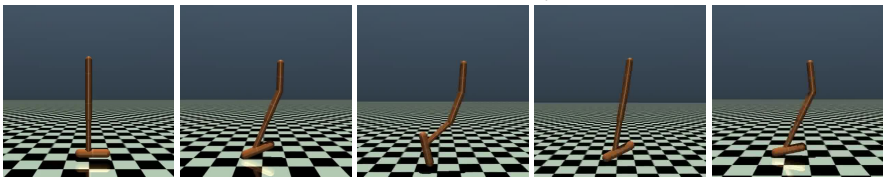
For ease of visualization, we show the normalized performance of the different algorithms in Figure 2. See Appendix B for raw absolute results. We also include another competitive baseline "GAIL-Expert-Noise" relevant to this scenario that artificially adds independent noise to the demonstration data for every gradient update during GAIL training. For a very high noise level, any



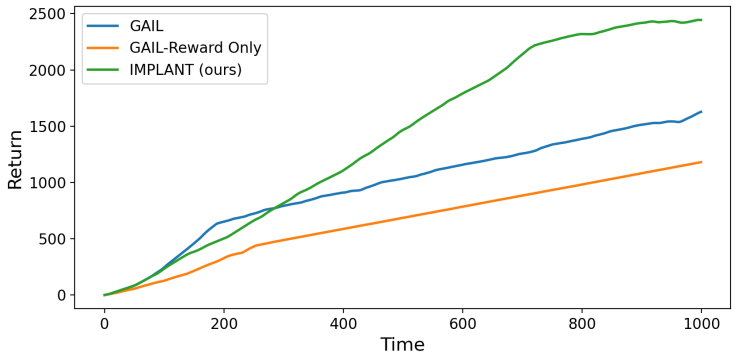
(a) GAIL



(b) GAIL-Reward Only



(c) IMPLANT (ours)



(d) Return over time of the above agents

Figure 1: Trajectory visualization for Hopper environment in causal confusion setup at test-time. While all agents start from the same state, only IMPLANT can effectively hop forward. All agents are trained in the confounded setting and tested in the non-confounded setting.

algorithm will naturally deteriorate in performance due to significant shift in training and testing environments. More importantly, for modest noise levels, we find that IMPLANT outperforms the baselines in almost all cases, highlighting its robustness.

4.4 Effect of Planning Horizon

Finally, we analyze the effect of planning horizon on IMPLANT performance in the same setup as Section 4.1. Specifically, we vary the planning horizon $H \in [0, 10, 50, 100]$ for a rollout budget $B = 10$. The normalized performance curves are shown in Figure 3. When the planning horizon is 0, we only rely on the terminal value function for estimating returns. Conversely, for large planning horizons (*e.g.*, $H = 100$), the returns are dominated by rewards accumulated at every time step. We observe that picking neither a very large horizon ($h \geq 100$) nor a very small one ($h = 0$) results in optimal performance, suggesting imperfections in both the

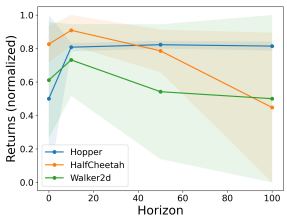
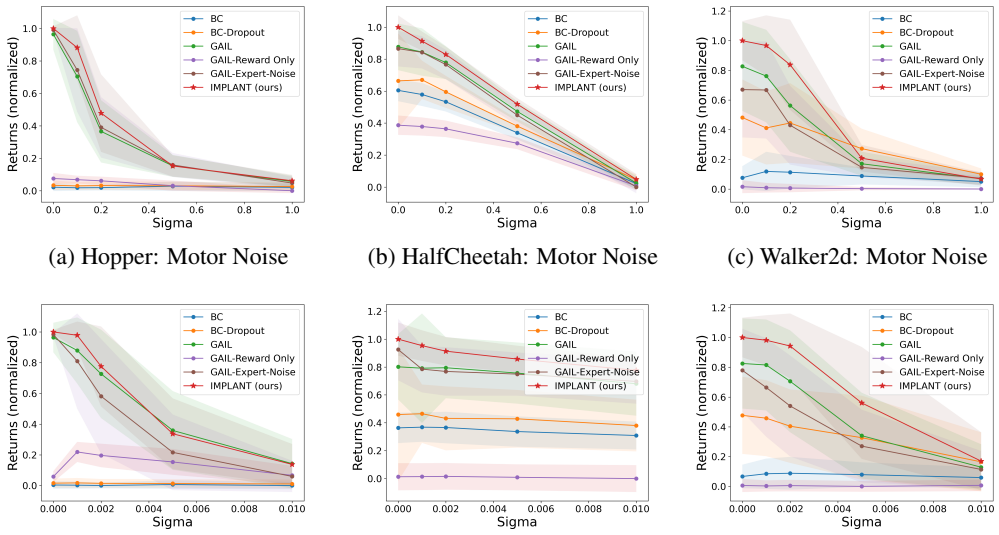


Figure 3: Effect of varying planning horizon H on IMPLANT performance.



(a) Hopper: Motor Noise (b) HalfCheetah: Motor Noise (c) Walker2d: Motor Noise

(d) Hopper: Transition Noise (e) HalfCheetah: Transition Noise (f) Walker2d: Transition Noise

Figure 2: Average return of imitation learning algorithms on motor and transition noise settings.

learned reward and value functions and the sweet-spot for the planning horizon is typically between the extremes.

5 Discussion & Related Work

Traditionally, algorithms for imitation learning fall into one of two categories. They are either completely model-free during both training and execution, as in behavioral cloning and its variants (Pomerleau, 1991; Ross et al., 2011). Alternatively, they are model-based in the sense that they utilize dynamics and (inferred) rewards models during training, but are model-free during execution, as in inverse reinforcement learning (Ng et al., 2000; Ratliff et al., 2006; Ziebart et al., 2008). Our work introduces a novel model-based perspective to imitation learning where the reward and transition models are used *both* during training and execution. Borrowing the terminology from Sutton & Barto (2018), the use of models for the MDP during training and execution are also referred to as *background* and *decision-time* planning respectively.

While imitation via background planning has showed immense promise for control in complex environments (Abbeel & Ng, 2004; Ratliff et al., 2009; Ho & Ermon, 2016; Choudhury et al., 2018), we showed that decision-time planning in IMPLANT can further improve the data efficiency and robustness of the learned policies. There have also been several alternate attempts for characterizing and enhancing the robustness of imitation policies. For example, Fu et al. (2017) seek robustness in the sense of recovering the true reward function via adversarial imitation learning and transfer the inferred reward function to external dynamics in the non-zero shot setting. A significant body of work also considers IRL approaches that can accurately capture the uncertainty in the reward function for safe deployment (Zheng et al., 2014; Brown et al., 2018; Huang et al., 2018; Lacotte et al., 2019; Brown et al., 2020). While these utility-based notions are distinct from ours, they are complementary approaches to robustness that could be combined with IMPLANT in future work.

Given the synergies between generative modeling and imitation learning as exemplified in GAIL (Ho & Ermon, 2016), improvements in the former often translate into improved imitation, *e.g.*, the use of autoencoder embeddings to improve diversity (Wang et al., 2017), better loss functions and architectures for stable GAN/GAIL training (Pfau & Vinyals, 2016; Kuefler et al., 2017; Li et al., 2017), etc. These modifications are conceptually complementary to the key contribution of IMPLANT to incorporate decision-time planning and are likely to further boost our performance. In fact, decision-time planning in IMPLANT can be viewed as filtering of trajectories sampled from the policy network. This is similar to recent work in using importance weighting for improving sample quality of a

generative model (Grover & Ermon, 2017; Azadi et al., 2018; Grover et al., 2019). However, our solution is tailored towards sequential decision making and deterministically picks the best outcome in line with model predictive control, unlike importance weighting filters.

6 Conclusion

We presented *Imitation with Planning at Test-time* (IMPLANT), a new algorithm for imitation learning that uses decision-time planning to mitigate compounding errors of any base IRL algorithm. Unlike existing approaches, IMPLANT is truly model-based in the sense of utilizing the inferred rewards and dynamics model both during training and execution. We demonstrated that IMPLANT matches or outperforms existing benchmark imitation learning algorithms with very few expert trajectories. Finally, we empirically demonstrated the robustness of IMPLANT via its impressive performance at zero-shot generalization in several challenging perturbation settings.

References

Pieter Abbeel and Andrew Y Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, pp. 1, 2004.

Samaneh Azadi, Catherine Olsson, Trevor Darrell, Ian Goodfellow, and Augustus Odena. Discriminator rejection sampling. *arXiv preprint arXiv:1810.06758*, 2018.

Nir Baram, Oron Anschel, and Shie Mannor. Model-based adversarial imitation learning. *arXiv preprint arXiv:1612.02179*, 2016.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

Daniel S Brown, Yuchen Cui, and Scott Niekum. Risk-aware active inverse reinforcement learning. In *Conference on Robot Learning*, pp. 362–372, 2018.

Daniel S Brown, Scott Niekum, and Marek Petrik. Bayesian robust optimization for imitation learning. *arXiv preprint arXiv:2007.12315*, 2020.

Eduardo F Camacho and Carlos Bordons Alba. *Model predictive control*. Springer Science & Business Media, 2013.

Sanjiban Choudhury, Mohak Bhardwaj, Sankalp Arora, Ashish Kapoor, Gireeja Ranade, Sebastian Scherer, and Debadeepta Dey. Data-driven planning via imitation learning. *The International Journal of Robotics Research*, 37(13-14):1632–1672, 2018.

Paul Christiano, Zain Shah, Igor Mordatch, Jonas Schneider, Trevor Blackwell, Joshua Tobin, Pieter Abbeel, and Wojciech Zaremba. Transfer from simulation to real world through learning deep inverse dynamics model. *arXiv preprint arXiv:1610.03518*, 2016.

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*, pp. 4754–4765, 2018.

Pim de Haan, Dinesh Jayaraman, and Sergey Levine. Causal confusion in imitation learning. In *Advances in Neural Information Processing Systems*, pp. 11698–11709, 2019.

Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, Yuhuai Wu, and Peter Zhokhov. Openai baselines. `https://github.com/openai/baselines`, 2017.

Justin Fu, Katie Luo, and Sergey Levine. Learning robust rewards with adversarial inverse reinforcement learning. *arXiv preprint arXiv:1710.11248*, 2017.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pp. 2672–2680, 2014.

Aditya Grover and Stefano Ermon. Boosted generative models. *arXiv preprint arXiv:1702.08484*, 2017.

Aditya Grover, Jiaming Song, Ashish Kapoor, Kenneth Tran, Alekh Agarwal, Eric J Horvitz, and Stefano Ermon. Bias correction of learned generative models using likelihood-free importance weighting. In *Advances in Neural Information Processing Systems*, pp. 11058–11070, 2019.

Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In *Advances in neural information processing systems*, pp. 4565–4573, 2016.

Jessie Huang, Fa Wu, Doina Precup, and Yang Cai. Learning safe policies with expert guidance. In *Advances in Neural Information Processing Systems*, pp. 9105–9114, 2018.

Vijay R Konda and John N Tsitsiklis. Actor-critic algorithms. In *Advances in neural information processing systems*, pp. 1008–1014, 2000.

Alex Kuefler, Jeremy Morton, Tim Wheeler, and Mykel Kochenderfer. Imitating driver behavior with generative adversarial networks. In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pp. 204–211. IEEE, 2017.

Jonathan Lacotte, Mohammad Ghavamzadeh, Yinlam Chow, and Marco Pavone. Risk-sensitive generative adversarial imitation learning. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pp. 2154–2163. PMLR, 2019.

Yunzhu Li, Jiaming Song, and Stefano Ermon. Infogail: Interpretable imitation learning from visual demonstrations. In *Advances in Neural Information Processing Systems*, pp. 3812–3822, 2017.

Anusha Nagabandi, Gregory Kahn, Ronald S Fearing, and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 7559–7566. IEEE, 2018.

Andrew Y Ng, Stuart J Russell, et al. Algorithms for inverse reinforcement learning. In *Icml*, volume 1, pp. 2, 2000.

Takayuki Osa, Joni Pajarinen, Gerhard Neumann, J Andrew Bagnell, Pieter Abbeel, and Jan Peters. An algorithmic perspective on imitation learning. *arXiv preprint arXiv:1811.06711*, 2018.

Jan Peters and Stefan Schaal. Natural actor-critic. *Neurocomputing*, 71(7-9):1180–1190, 2008.

David Pfau and Oriol Vinyals. Connecting generative adversarial networks and actor-critic methods. *arXiv preprint arXiv:1610.01945*, 2016.

Dean A Pomerleau. Efficient training of artificial neural networks for autonomous navigation. *Neural computation*, 3(1):88–97, 1991.

Martin L Puterman. Markov decision processes. *Handbooks in operations research and management science*, 2:331–434, 1990.

Nathan D Ratliff, J Andrew Bagnell, and Martin A Zinkevich. Maximum margin planning. In *Proceedings of the 23rd international conference on Machine learning*, pp. 729–736, 2006.

Nathan D Ratliff, David Silver, and J Andrew Bagnell. Learning to search: Functional gradient techniques for imitation learning. *Autonomous Robots*, 27(1):25–53, 2009.

Arthur George Richards. *Robust constrained model predictive control*. PhD thesis, Massachusetts Institute of Technology, 2005.

Stéphane Ross and Drew Bagnell. Efficient reductions for imitation learning. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 661–668, 2010.

Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 627–635, 2011.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pp. 1889–1897, 2015.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033. IEEE, 2012.

Ziyu Wang, Josh S Merel, Scott E Reed, Nando de Freitas, Gregory Wayne, and Nicolas Heess. Robust imitation of diverse behaviors. In *Advances in Neural Information Processing Systems*, pp. 5320–5329, 2017.

Jiangchuan Zheng, Siyuan Liu, and Lionel M Ni. Robust bayesian inverse reinforcement learning with sparse behavior noise. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, pp. 2198–2205, 2014.

Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, and Anind K Dey. Maximum entropy inverse reinforcement learning. In *Aaai*, volume 8, pp. 1433–1438. Chicago, IL, USA, 2008.

Table 3: Detailed information on environments

Environment	Observation space	Action space
Hopper	Box(11,)	Box(3,)
HalfCheetah	Box(17,)	Box(6,)
Walker2d	Box(17,)	Box(6,)

Table 4: Detailed information on expert data used in training

Environment	# of trajectories	# of state-action pairs
Hopper	4	200
HalfCheetah	4	200
Walker2d	20	1000

A Additional Experimental Details and Setups

A.1 Environments and Expert Data

As mentioned above, we consider 3 continuous tasks from OpenAI Gym (Brockman et al., 2016) simulated with MuJoCo (Todorov et al., 2012): Hopper, HalfCheetah, and Walker2d. We acquire expert data from OpenAI Baselines (Dhariwal et al., 2017). Table 3 lists more detailed information about each environment, and Table 4 contains information about the expert demonstrations we use for training all of the agents.

A.2 Hyperparameters and Network Architectures

We use a 2-layer MLP with *tanh* activations and 64 hidden units for all of our policy networks. For BC, we use a learning rate of 10^{-4} across all environments. We use a dropout rate of 0.2 for our BC-Dropout agent. The hyperparameters of GAIL are listed in Table 5. For our IMPLANT agent, we directly utilize the value function, reward function, and policy from a trained GAIL agent. In all settings, we choose $B = 20$ and $H = 50$ for Hopper, $B = 2$ and $H = 10$ for HalfCheetah and Walker2d to plan.

B Additional Results

The complete results for Section 4.3 can be found in Table 6, 7, 8.

Table 5: Detailed information on GAIL's hyperparameters

Parameters	Hopper	HalfCheetah	Walker2d
Discriminator network	100-100 MLP	100-100 MLP	100-100 MLP
Discriminator entropy coeff.	0.01	0.01	0.01
Batch size	1024	1024	1024
Max kl	0.01	0.01	0.01
CG steps/damping	10, 0.01	10, 0.1	10, 0.1
Entropy coeff.	0.0	0.0	0.0
Value fn. steps/step size	3, 3e-4	5,1e-3	5,1e-3
Generator steps	3	3	3
Discriminator steps	1	1	1
λ	0.98	0.97	0.97
γ	0.99	0.995	0.995

Table 6: Raw results of Figure 2 in Hopper environment

(a) Hopper with motor noise

Sigma	0.0	0.1	0.2	0.5	1.0
BC	127 ± 85	114 ± 64	117 ± 61	179 ± 105	123 ± 82
BC-Dropout	169 ± 105	156 ± 85	163 ± 95	158 ± 89	142 ± 94
GAIL	3506 ± 337	2572 ± 1008	1360 ± 687	598 ± 243	252 ± 128
GAIL-Reward Only	319 ± 123	293 ± 87	268 ± 92	160 ± 113	49 ± 54
GAIL-Expert-Noise	3602 ± 46	2716 ± 921	1449 ± 667	**618** ± 264	218 ± 136
IMPLANT (ours)	**3633** ± 50	**3209** ± 714	**1764** ± 856	596 ± 234	**269** ± 130

(b) Hopper with transition noise

Sigma	0.0	0.001	0.002	0.005	0.01
BC	127 ± 85	123 ± 81	116 ± 65	137 ± 98	114 ± 84
BC-Dropout	169 ± 105	175 ± 113	165 ± 99	161 ± 98	151 ± 101
GAIL	3506 ± 337	3209 ± 756	2672 ± 1012	**1377** ± 901	**616** ± 563
GAIL-Reward Only	319 ± 123	884 ± 230	804 ± 269	665 ± 296	349 ± 215
GAIL-Expert-Noise	3576 ± 36	2966 ± 1089	2160 ± 1311	875 ± 865	323 ± 357
IMPLANT (ours)	**3633** ± 50	**3557** ± 313	**2844** ± 915	1301 ± 810	598 ± 467

Table 7: Raw results of Figure 2 in HalfCheetah environment

(a) HalfCheetah with motor noise

Sigma	0.0	0.1	0.2	0.5	1.0
BC	427 ± 131	377 ± 139	289 ± 118	-87 ± 81	-703 ± 66
BC-Dropout	542 ± 275	555 ± 232	409 ± 194	-6 ± 85	-655 ± 61
GAIL	954 ± 282	892 ± 287	764 ± 200	172 ± 121	-697 ± 82
GAIL-Reward Only	5 ± 117	-11 ± 110	-39 ± 100	-214 ± 72	-732 ± 53
GAIL-Expert-Noise	933 ± 216	888 ± 195	740 ± 179	127 ± 124	-745 ± 74
IMPLANT (ours)	**1193** ± 143	**1025** ± 115	**863** ± 94	**260** ± 72	**-652** ± 65

(b) HalfCheetah with transition noise

Sigma	0.0	0.001	0.002	0.005	0.01
BC	427 ± 131	433 ± 126	430 ± 136	395 ± 131	361 ± 120
BC-Dropout	542 ± 275	549 ± 252	509 ± 276	505 ± 249	447 ± 223
GAIL	954 ± 282	942 ± 472	946 ± 263	900 ± 264	808 ± 276
GAIL-Reward Only	5 ± 117	6 ± 116	8 ± 115	$0 \pm 112.$	-11 ± 116
GAIL-Expert-Noise	1103 ± 265	934 ± 201	914 ± 209	891 ± 186	828 ± 192
IMPLANT (ours)	**1193** ± 143	**1137** ± 132	**1089** ± 123	**1021** ± 115	**923** ± 103

Table 8: Raw results of Figure 2 in Walker2d environment

(a) Walker2d with motor noise

Sigma	0.0	0.1	0.2	0.5	1.0
BC	258 ± 262	402 ± 444	384 ± 368	298 ± 191	172 ± 128
BC-Dropout	1622 ± 861	1386 ± 826	1498 ± 901	**918** ± 450	**339** ± 129
GAIL	2780 ± 1007	2560 ± 1047	1893 ± 1045	576 ± 272	234 ± 133
GAIL-Reward Only	56 ± 146	33 ± 109	25 ± 94	13 ± 36	4 ± 13
GAIL-Expert-Noise	2253 ± 1081	2245 ± 1098	1448 ± 891	495 ± 221	247 ± 144
IMPLANT (ours)	**3360** ± 442	**3251** ± 682	**2816** ± 1019	701 ± 301	228 ± 148

(b) Walker2d with transition noise

Sigma	0.0	0.001	0.002	0.005	0.01
BC	258 ± 262	318 ± 338	329 ± 365	298 ± 274	233 ± 230
BC-Dropout	1622 ± 861	1559 ± 848	1380 ± 819	1123 ± 937	584 ± 658
GAIL	2780 ± 1007	2746 ± 1015	2383 ± 1136	1168 ± 958	464 ± 511
GAIL-Reward Only	56 ± 146	46 ± 124	53 ± 128	36 ± 120	59 ± 129
GAIL-Expert-Noise	2627 ± 927	2244 ± 1098	1832 ± 1125	935 ± 812	414 ± 412
IMPLANT (ours)	**3360** ± 442	**3299** ± 558	**3170** ± 725	**1899** ± 1249	**600** ± 644