

# RAPIDLY ADAPTING POLICIES TO THE REAL WORLD VIA SIMULATION-GUIDED FINE-TUNING

Patrick Yin<sup>\*,1</sup>, Tyler Westenbroek<sup>\*1</sup>,  
Ching-an Cheng<sup>2</sup>, Andrey Kobolov<sup>2</sup> & Abhishek Gupta<sup>1</sup>

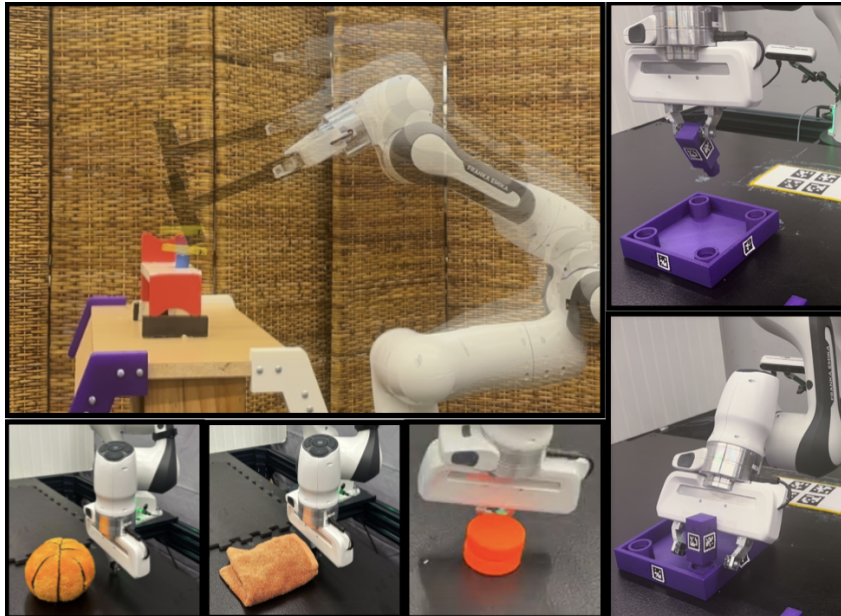


Figure 1: Five dynamic, contact-rich manipulation tasks – hammering (**top left**), insertion (**right**), and three pushing (**bottom left**) tasks – solved in the real world using *SGFT*.

## ABSTRACT

Robot learning requires a considerable amount of high-quality data to realize the promise of generalization. However, large data sets are costly to collect in the real world. Physics simulators can cheaply generate vast data sets with broad coverage over states, actions, and environments. However, physics engines are fundamentally misspecified approximations to reality. This makes direct zero-shot transfer from simulation to reality challenging, especially in tasks where precise and force-sensitive manipulation is necessary. Thus, fine-tuning these policies with small real-world data sets is an appealing pathway for scaling robot learning. However, current reinforcement learning fine-tuning frameworks leverage general, unstructured exploration strategies which are too inefficient to make real-world adaptation practical. This paper introduces the *Simulation-Guided Fine-tuning* (SGFT) framework, which demonstrates how to extract structural priors from physics simulators to substantially accelerate real-world adaptation. Specifically, our approach uses a value function learned in simulation to guide real-world exploration. We demonstrate this approach across five real-world dexterous manipulation tasks where zero-shot sim-to-real transfer fails. We further demonstrate our framework substantially outperforms baseline fine-tuning methods, requiring up to an order of magnitude fewer real-world samples and succeeding at difficult tasks where prior approaches fail entirely. Last but not least, we provide theoretical justification for this new paradigm which underpins how SGFT can rapidly learn high-performance policies in the face of large sim-to-real dynamics gaps. Project webpage: [weirdlabuw.github.io/sgft](https://weirdlabuw.github.io/sgft)

<sup>\*</sup>Equal contribution 1. University of Washington 2. Microsoft Research

## 1 INTRODUCTION

Robot learning offers a pathway to building robust, general-purpose robotic agents that can rapidly adapt their behavior to new environments and tasks. This shifts the burden from designing task-specific controllers by hand to the problem of collecting large behavioral datasets with sufficient coverage. This raises a fundamental question: how do we cheaply obtain and leverage such datasets at scale? Real-world data collection via teleoperation (Walke et al., 2023; team, 2024) can generate high-quality trajectories but scales linearly with human effort. Even with community-driven teleoperation (Mandlekar et al., 2018; Collaboration, 2024), current robotics datasets are orders of magnitude smaller than those powering vision and language applications.

Massively parallelized physics simulation (Todorov et al., 2012; Makoviychuk et al., 2021) can cheaply generate vast synthetic robotics data sets. Indeed, generating data sets with extensive coverage over environments, states, and actions can be largely automated using techniques such as automatic scene generation (Chen et al., 2024; Deitke et al., 2022), dynamics randomization (Peng et al., 2018; Andrychowicz et al., 2020), and search algorithms such as reinforcement learning.

Unfortunately, simulation-generated data is not a silver bullet, as it provides cheap but ultimately off-domain data. Namely, simulators are fundamentally misspecified approximations to reality. This is highlighted in tasks like hammering in a nail, where the modeling of high-impact, deformable contact remains an open problem (Acosta et al., 2022; Levy et al., 2024). In these regions, choice of parameters for the physics simulator accurately capture the real-world dynamics. This gap persists despite efforts towards improving existing physics simulators with system identification (Memmel et al., 2024; Huang et al., 2023; Levy et al., 2024). Thus, despite impressive performance for many tasks, methods that transfer policies from simulation to reality zero-shot (Kumar et al., 2021; Lee et al., 2020; Peng et al., 2018; Andrychowicz et al., 2020) run into failure modes when they encounter novel dynamics not covered by the simulator (Smith et al., 2022b).

The question becomes: can inaccurate simulation models be useful in the face of fundamental misspecifications? A natural technique is to fine-tune policies pre-trained in a simulator using real-world experience (Smith et al., 2022b; Zhang et al., 2023). This approach can overcome misspecification by training directly on data from the target domain. However, existing approaches typically employ the unstructured exploration strategies used by standard, general reinforcement learning algorithms Haarnoja et al. (2018). As a result, current RL fine-tuning frameworks remain too sample-inefficient for real-world deployment. We argue the following: despite getting the finer details wrong, physics simulators capture the rough structure of real-world dynamics well enough to transfer targeted exploration and adaptation strategies from simulation to reality.

This motivates the Simulation-Guided Fine-Tuning (SGFT) framework, which uses a value function  $V_{\text{sim}}$  learned in the simulator to transfer behavioral priors from simulation to reality. Our key insight is that the ordering defined by  $V_{\text{sim}}$  captures successful behaviors – such as reaching towards and object, picking it, and moving it to a desired position – which are invariant across simulation and reality, even if the low-level dynamics diverge substantially in the two domains. In more detail, SGFT departs from standard fine-tuning strategies by using  $V_{\text{sim}}$  to synthesize dense rewards to guide targeted real-world exploration and shortening the learning horizon when adapting in the real-world. Prior approaches Smith et al. (2022b); Zhang et al. (2023) optimize the same finite-horizon objective in both simulation and reality, and thus use the simulator to initialize real-world learning. These approaches often suffer from catastrophic forgetting Wojczyk et al. (2024), where there is a substantial drop in policy performance early in the fine-tuning phase. Incorporating  $V_{\text{sim}}$  into the reward enables the agent to retain structural information about the optimal behaviors learned in simulation throughout the fine-tuning processes, while shorter learning horizons are well-known to yield more trackable policy optimization problems Westenbroek et al. (2022); Cheng et al. (2019). Altogether, SGFT provides a stronger learning signal which enables base policy optimization algorithms to consistently and rapidly improve real-world performance.

Although SGFT is a very general framework for sim-to-real fine-tuning, we place a special emphasis on implementations which use sample-efficient model-based reinforcement learning algorithms Janner et al. (2019); Hansen et al. (2024). Model-based approaches typically struggle with long-horizon tasks due to compounding errors in model predictions Janner et al. (2019), but our horizon-shortening strategy conveniently side-steps this challenge, fully unlocking the performance benefits promised by generative world models. We outline our contributions as follows:

Figure 2: Depiction of a model-based instantiation of SGFT. While prior approaches optimize the same challenging in finite-horizon objective during simulation pretraining and real-world fine-tuning, SGFT modifies the fine-tuning objective by a) using the value function learned in simulation ( $v_{sim}$ ) to reshape rewards and guide efficient real-world exploration, and b) shortening the search horizon to make real-world learning more tractable. For the model-based instantiation depicted (and outlined in Section 4.2), this amounts to branching short, exploratory model-based rollouts from real-world trajectories to search for sequences of actions which will improve policy performance. Intuitively, SGFT uses the simulator to approximately bootstrap long-horizon rewards (via  $v_{sim}$ ), while small amounts of real-world data are used to search for short sequences of optimal actions under the real-world dynamics.

1. We introduce the SGFT framework, which requires modifying only a few lines of code in existing sim-to-real RL fine-tuning frameworks Smith et al. (2022b); Zhang et al. (2019) and can be built on top of any base policy optimization algorithm.
2. We implement two model-based instantiations of SGFT and demonstrate through five contact-rich, sim-to-real robot manipulation experiments that SGFT provides substantial sample complexity gains over existing fine-tuning methods.
3. We demonstrate theoretically how SGFT can learn highly performant policies, despite the bias introduced by the SGFT objective and b) how this enables SGFT to overcome compounding errors which plague standard MBRL approaches.

Together, these insights underscore how SGFT effectively leverages plentiful off-domain data from a simulator alongside small real-world data sets to substantially accelerate real-world fine-tuning.

## 2 RELATED WORK

**Simulation-to-Reality Transfer.** Two main approaches have been proposed for overcoming the dynamics gap between simulation and reality: 1) adapting simulation parameters to real-world data (Chebotar et al., 2019; Ramos et al., 2019; Memmel et al., 2024) and 2) learning adaptive or robust policies to account for uncertain real-world dynamics (Qi et al., 2022; Kumar et al., 2021; Yu et al., 2017). However, these approaches still display failure modes in regimes where simulation and reality diverge substantially Smith et al. (2022a) – namely, where no set of simulation parameters closely match the real-world dynamics.

**Adapting Policies with Real-World Data.** Many RL approaches have focused on the general fine-tuning problem (Rajeswaran et al., 2018; Nair et al., 2020; Kostrikov et al., 2021; Hu et al., 2023; Nakamoto et al., 2024). These works initialize policies, Q-functions, and replay buffers from offline data and continue training them with standard RL methods. A number of works have specifically considered mixing simulated and real data during policy optimization – either through co-training (Torne et al., 2024), simply initializing the replay-buffer with simulation data (Smith et al., 2022a; Ball et al., 2023), or adaptively sampling the simulated dataset and up-weighting transitions that approximately match the true dynamics (Eysenbach et al., 2020; Liu et al., 2022; Xu et al., 2023; Niu et al., 2022). However, recent work Zhou et al. (2024) has demonstrated that there are minimal benefits to sampling off-domain samples when adapting online in new environments, as this can bias learned policies towards sub-optimal solutions. In contrast to these prior approaches – which primarily use simulated experience to initialize real-world learning – we focus on distilling effective exploration strategies from simulation, using  $v_{sim}$  to guide real-world learning. We demonstrate theoretically that this approach to transfer leads to low bias in the policies learned by SGFT.

**Model-Based Reinforcement Learning.** A significant body of work on model-based RL learns a generative dynamics models to accelerate policy optimization (Sutton, 1991; Wang & Ba, 2019; Janner et al., 2019; Yu et al., 2020; Kidambi et al., 2020; Ebert et al., 2018; Zhang et al., 2019). In principle, a model enables the agent to make predictions about states and actions not contained in the training data, enabling rapid adaptation to new situations. However, the central challenge for model-based methods is that small inaccuracies in predictive models can quickly compound over time, leading to large model-bias and a drop in controller performance. An effective critic can be used to shorten search horizons (Hansen et al., 2024; Bhardwaj et al., 2020; Hafner et al., 2019; Jadbabaie et al., 2001; Grune & Rantzer, 2008) yielding easier decision-making problems, but learning such a critic from scratch can still require large amounts of on-task data. We demonstrate that, for many real-world continuous control problems, critics learned entirely in simulation can be robustly transferred to the real-world and substantially accelerate model-based learning.

**Reward Design in Reinforcement Learning.** A significant component of our methodology is learning dense shaped reward in simulation to guide real-world fine-tuning. Prior techniques have tried to infer rewards from expert demos (Ziebart et al., 2008; Ho & Ermon, 2016), success examples (Fu et al., 2018; Li et al., 2021), LLMs (Ma et al., 2023; Yu et al., 2023), and heuristics (Margolis et al., 2024; Berner et al., 2019). We rely on simulation to provide reward supervision Westebroek et al. (2022) using the PBRs formalism (Ng et al., 1999). This effectively encodes information about the dynamics and optimal behaviors in the simulator, enabling us to shorten the learning horizon and improve sample efficiency Cheng et al. (2021); Westebroek et al. (2022).

### 3 PRELIMINARIES

Let  $S$  and  $A$  be state and action spaces. Our goal is to control a real-world system defined by an unknown Markovian dynamics  $p_{\text{real}}(s'; s, a)$ , where  $s^0 \in S$  are states and  $a \in A$  is an action. The usual formalism for solving tasks with RL is to define a Markov Decision Process of the form  $M_r = (S; A; p_{\text{real}}; r; \gamma)$  with initial real-world state distribution  $\mu_{\text{real}}^0$ , reward function  $r$ , and discount factor  $\gamma \in [0, 1)$ . Given a policy  $\pi$ , we let  $d_{\text{real}}(s)$  denote the distribution over trajectories  $\tau = (s_0; a_0; s_1; a_1; \dots)$  generated by applying starting at initial state  $s_0$ . Denoting the value function under  $\pi$  as  $V_{\text{real}}(s) = E_{s \sim \mu_{\text{real}}^0, \tau \sim d_{\text{real}}(s)}[\sum_{t=0}^{\infty} \gamma^t r_t(s_t)]$ , our objective is to find  $\pi_{\text{real}}^*$  such that  $V_{\text{real}}^*(s) := \sup_{\pi} V_{\text{real}}(s)$ .

Unfortunately, obtaining a good approximation to  $\pi_{\text{real}}^*$  using only real-world data is often impractical. Thus, many approaches leverage an approximate simulation environment  $M_{\text{sim}} := (S; A; p_{\text{sim}}; r_{\text{sim}}; \gamma)$  and solve an approximate MDP of the form  $M_{\text{sim}}$  to train a policy  $\pi_{\text{sim}}$  meant to approximate  $\pi_{\text{real}}^*$ . We let  $V_{\text{sim}}$  denote  $\pi_{\text{sim}}$ 's value function with respect to  $\mu_{\text{sim}}^0$ . Here,  $\mu_{\text{sim}}^0$  is the distribution over initial conditions in the simulator.

### 4 SIMULATION-GUIDED FINE-TUNING

We build our framework around the following intuition: even when the finer details of  $p_{\text{real}}$  and  $p_{\text{sim}}$  differ substantially, we can often assume that  $\pi_{\text{sim}}$  captures the rough motions needed to complete a task in the real world (such as swinging a hammer towards a nail). Specifically, we hypothesize that the ordering defined by  $V_{\text{sim}}$  captures these behaviors in a form that can be robustly transferred from simulation to reality and used to guide efficient real-world adaptation.

#### 4.1 SIMULATION-GUIDED FINE-TUNING

When fine-tuning  $\pi_{\text{sim}}$  to the real world we propose (a) reshaping the original reward function according to  $r(s; s^0) = r(s) + \beta (V_{\text{sim}}(s^0) - V_{\text{sim}}(s))$  and (b) shortening the search horizon to a more tractable  $H$ -step objective  $\sum_{t=0}^{H-1} \gamma^t r(s_t; s_{t+1})$ . The reshaped objective is an instance of the Potential-Based Reward Shaping (PRBS) formalism (Ng et al., 1999), where  $V_{\text{sim}}$  is used as the potential function. This reshaping approach is typically applied to finite-horizon objectives, where it can be shown that the optimal policy under  $r(s; s^0)$  is the same as the optimal policy under the original reward  $r(s)$  (Ng et al., 1999). In contrast, the infinite horizon search problem biases the objective towards behaviors which were successful in the simulator. Indeed, by telescoping out terms we can rewrite our policy optimization objective as:

$$V_H^*(s) = E_{s_0 \sim \mu_{\text{sim}}^0} [V_{\text{sim}}(s_H) + \sum_{t=0}^{H-1} \gamma^t r(s_t) + \sum_{t=0}^{\infty} \gamma^t r(s_t) - V_{\text{sim}}(s_0) | s_0 = s; a_t = \pi_{\text{sim}}(s_t); \quad (1)$$

$$V_H(s) := \sup_{\pi} V_{\pi}^H(s); \quad Q_H(s; \cdot) := E_{a \sim (\cdot|s)} V_{\pi}^H(s^0) + r(s) :$$

Here  $\pi = \{ \pi_0; \pi_1; \dots; \pi_{H-1} \}$  denotes  $H$ -step time-dependent policies, where  $\pi_t$  is the policy applied at time  $t$ . We emphasize that these step returns are optimized over real-world dynamics. We propose learning a policy which optimizes these step returns from each state:

$$Q_H(s; \cdot) := \sup_{\pi} Q_{\pi}^H(s; \cdot) :$$

What behavior does this objective elicit? For each  $H$ , the  $H$ -step Bellman equation dictates that the optimal policy  $\pi_{\text{real}}^H$  for the original MDP  $\mathcal{M}_{\text{real}}$  can be found by solving:

$$Q_{\pi_{\text{real}}^H}(s; \cdot) := \sup_{\pi} E_{\pi} [ V_{\pi}^H(s_H) + \sum_{t=0}^{H-1} \gamma^t r(s_t) | V_{\pi_{\text{real}}^H}(s_0) ] \quad s_0 = s; a_t \sim (\cdot|s_t) :$$

Thus, (1) effectively uses  $V_{\text{sim}}$  as a surrogate for  $V_{\text{real}}$ . Namely, (1) uses the simulator to bootstrap long-horizon returns, while real-world interaction is optimized only over short trajectory segments. In the extreme case where  $\gamma = 1$ ,  $\pi_{\text{real}}^H$  will greedily attempt to increase  $V_{\text{sim}}$  at each timestep; namely, the policy search problem will be reduced to a contextual bandit problem. As we take  $H \rightarrow 1$ ,  $\pi_{\text{real}}^H$  will optimize purely real long-horizon returns and thus recover the behavior of  $\pi_{\text{real}}$ .

We are particularly interested in optimizing this objective with small values of  $\gamma$ . This provides an ideal separation between what is learned using cheap, plentiful interactions with  $\mathcal{M}_{\text{sim}}$  and what is learned with more costly interactions with  $\mathcal{M}_{\text{real}}$ . In simulation, we can easily generate enough data to explore many paths through the state space and discover which motions lead to higher returns (e.g. swinging a hammer towards a nail). This information is distilled into  $V_{\text{sim}}$  during the learning process, which defines an ordering over which states are more desirable to visit in the future. By optimizing the reshaped objective Equation (1) for small values of  $\gamma$ , we only need to learn short sequences of actions which move the real-world system to states where  $V_{\text{sim}}$  is higher. Intuitively, (1) learns where to go with large amounts of simulated data and how to get there with small amounts of real-world data, efficiently adapting  $\pi_{\text{sim}}$  to the real-world dynamics.

**Connections to the MPC Literature:** Note that the  $V_{\text{sim}}(s_0)$  term in (1) does not depend on the choice of policy, and thus does not affect the choice of optimal policy. Thus, (1) is equivalent to the planning objective used by model predictive control (MPC) methods (Jadbabaie et al., 2001; Hansen et al., 2024; Sun et al., 2018; Bhardwaj et al., 2020) with a step look-ahead and a terminal reward of  $V_{\text{sim}}$  (assuming oracle access to the real-world dynamics). Of course, we cannot calculate the optimal MPC policy  $\pi_{\text{real}}^H$  directly because we do not know  $\mathcal{M}_{\text{real}}$ , and thus seek to approximate its behavior by learning from real-world interactions.

**The Simulation-Guided Fine-Tuning Training Loop.** We propose the general Simulation-Guided Fine-Tuning (SGFT) framework in

---

#### Algorithm 1 Simulation-Guided Fine-tuning (SGFT)

---

the pseudo-code in Algorithm 1. Require: Pretrained policy  $\pi_{\text{sim}}$  and value function  $V_{\text{sim}}$ . SGFT fine-tunes  $\pi_{\text{sim}}$  to succeed under the real-world dynamics by iteratively 1) unrolling the current policy to collect transitions from  $\mathcal{M}_{\text{real}}$  and 2) using the current dataset  $\mathcal{D}$  of transitions to approximately optimize  $\max_{\pi} Q_{\pi}^H(s; (\cdot|s_j))$  at each state  $s_j$  the agent has visited. By optimizing (1), SGFT optimizes policies towards the actions taken by  $\pi_{\text{sim}}$ . Note that this framework can be built on top of any base policy optimization method; however, over the next two sections we will argue that this framework is particularly beneficial when paired with model-based search strategies.

## 4.2 LEVERAGING SHORT MODEL ROLL-OUTS

Learning a generative model for  $\mathcal{M}_{\text{real}}$  with the real-world data set enables an agent to generate synthetic rollouts and reason about trajectories not contained in the data set. In principle, this should

substantially accelerate the learning of effective policies. The central challenge for model-based reinforcement learning is that small errors can quickly compound over multiple steps, degrading the quality of the predictions Janner et al. (2019). As a consequence, learning a model accurate enough to solve long-horizon problems can often take as much data as solving the task with modern model-free methods (Chen et al., 2021; Hiraoka et al., 2021). By bootstrapping in simulation, where data is plentiful, the SGFT framework enables agents to act effectively over long horizons using only short, local H-step predictions about the real-world dynamics. This side-steps the core challenge for model-based methods, and enables extremely efficient real-world learning. We now outline how the SGFT can be built on top of two predominant classes of MBRL algorithms.

Notation. In what follows, we will use  $\hat{\cdot}$  notation to denote H-step returns of the form (1) under the transitions generated by the model rather than the real dynamics. Namely,  $\hat{V}_H^{\pi}$  is the H-step value of policy  $\pi$  under  $\hat{p}$ ;  $\hat{V}_H^*$  and  $\hat{Q}_H^*$  are the optimal values. Similarly,  $\hat{\pi}_H^*$  denotes the optimal H-step policy under the model dynamics. Note that this is simply the MPC policy generated by using  $\hat{p}$  and optimizing (1) with  $\hat{p}$  substituted in for  $p_{\text{real}}$ .

---

Algorithm 2 Dyna-SGFT

---

Require: Pretrained policy  $\pi_{\text{sim}}$  and value function  $V_{\text{sim}}$

- 1:  $D \leftarrow \emptyset$
- 2: for each iteration  $k$  do
- 3:   Generate rollout  $(s_t; a_t; r_t; s_{t+1})_{t=0}^T$  under  $\pi_{\text{sim}}$ .
- 4:    $r_t \leftarrow r_t + V_{\text{sim}}(s_{t+1}) - V_{\text{sim}}(s_t)$
- 5:    $D \leftarrow D \cup \{(s_t; a_t; r_t; s_{t+1})\}$
- 6:   Fit generative model  $\hat{p}$  with  $D$ .
- 7:   for G policy updates do
- 8:     Generate synthetic branched rollouts under  $\hat{p}$ .
- 9:     Approx. optimize  $\max_{\pi} Q_H(\pi; (s_j))_{j \in \mathcal{S}}$
- 10:      $\mathcal{S} \leftarrow \mathcal{S} \cup D$  using augmented dataset  $D$
- 11:   end for

---

Improved Sample Efficiency with Data Augmentation (Algorithm 2). The generative model can be used for data augmentation by generating a dataset of synthetic rollouts to supplement the real-world dataset  $D$  (Janner et al., 2019; Sutton, 1990; Gu et al., 2016). The combined dataset can then be fed to any policy optimization strategy, such as generic model-free algorithms. We consider state-of-the-art Dyna-style algo-

---

Algorithm 3 MPC-SGFT

---

Require: Pretrained value  $V_{\text{sim}}$  and initialized model  $\hat{p}$ .

- 1: for each iteration  $k$  do
- 2:   Generate rollout  $(s_t; a_t; r_t; s_{t+1})_{t=0}^T$  using  $\hat{p}$  and trajectory optimization to calculate  $\hat{Q}_H$
- 3:    $r_t \leftarrow r_t + V_{\text{sim}}(s_{t+1}) - V_{\text{sim}}(s_t)$
- 4:    $D \leftarrow D \cup \{(s_t; a_t; r_t; s_{t+1})\}$
- 5:   Fit generative model  $\hat{p}$  with  $D$ .
- 6: end for

---

rithms (Janner et al., 2019), which, in our context, branch H-step rollouts from states the agent has visited previously in the real-world. As Algorithm 2 shows, after each data-collection phase, this approach updates  $\hat{p}$  then repeatedly a) generates a dataset of synthetic H-step rollouts under the current policy starting from states in  $D$ , and b) approximately solves  $\max_{\pi} Q_H(\pi; (s_j))_{j \in \mathcal{S}}$  at the observed real-world states using the augmented dataset  $D$  and a base model-free method such as SAC (Haarnoja et al., 2018).

Online Planning (Algorithm 3). The most straightforward way to approximate the behavior of  $\hat{\pi}_H$  is simply to apply the MPC controller  $\hat{\pi}_H$  generated using the current best guess for the dynamics  $\hat{p}$ . Algorithm 3 provides general pseudocode for this approach, which iteratively rolls out  $\hat{\pi}_H$  (which is calculated using online optimization and Williams et al., 2017)) then updates the model on the current dataset of transitions. This high-level approach encompasses a wide array of methods proposed in the literature, e.g., Ebert et al. (2018) and Zhang et al. (2019). For the experiments in Section 6, we implemented this approach using the TDMPC-2 (Hansen et al., 2024) algorithm, which additionally learns a policy prior to accelerate the trajectory optimization.

## 5 THEORETICAL ANALYSIS

The preceding discussions have covered how the dense rewards and horizon shortening strategy employed by SGFT can lead to efficient real-world adaptation. However this leaves the outstanding question: how does the bias introduced by the reward shaping and horizon shortening affect policy performance? Longer prediction horizons decrease the bias of the objective by relying more heavily on real returns, but lead to less sample efficient adaptation Cheng et al. (2021); Westenbroek et al.

(2022). Given these competing challenges, can we expect to simultaneously learn high-performing policies and adapt rapidly in the real-world?

We introduce a novel geometric analysis which demonstrates that we can achieve both goals, even when there is a large gap between  $p_{\text{sim}}$  and  $p_{\text{real}}$ . Specifically, we introduce mild technical conditions on the structure of  $p_{\text{sim}}$ ,  $V_{\text{sim}}$  and  $p_{\text{real}}$  which ensure that  $\pi_{\text{H}}$  is nearly optimal for  $M_{\text{real}}$ , even when short prediction horizons are used. Our analysis builds on the following insight: even when there is a large gap in the magnitude of the transition probabilities between simulation and reality, it is reasonable to assume that  $\pi_{\text{sim}}$  still defines a reasonable ordering over desirable states under the real-world dynamics  $p_{\text{real}}$ . Namely, we argue that  $V_{\text{sim}}$  can capture the structure of motions that complete the desired task (such as reaching towards and object, picking it up, and moving it to a desired location), even if the low-level sequences of actions needed to realize these motions differs substantially between simulation and reality (due e.g. to different contact dynamics). We use the following formal definition to capture this intuition, which is from Cheng et al. (2019) and shares strong connections to Lyapunov Westenbroek et al. (2022); Grune & Rantzer (2008) and Dissipation Brogliato et al. theories from dynamical systems and control:

Definition 1. We say that  $V_{\text{sim}}$  is improvable with respect to  $M_{\text{real}}$  if for each  $\epsilon > 0$  we have:

$$\max_a E_{s^0} [V_{\text{sim}}(s^0)] - V_{\text{sim}}(s) - \epsilon r(s) > 0 \quad (2)$$

To unpack why this definition is useful, note that  $V_{\text{sim}}$  is by definition improvable with respect to  $M_{\text{sim}}$ . Indeed, by the temporal difference equation we have  $E_{s^0} [V_{\text{sim}}(s^0)] - V_{\text{sim}}(s) = r(s)$ . Namely,  $V_{\text{sim}}$  is constructed so that policies can greedily increase  $V_{\text{sim}}$  under  $p_{\text{sim}}$  at each step and be guaranteed to reach desirable states in the long run, namely, states which correspond to successfully completing the task. Definition 1 ensures that this condition then holds under the real-world dynamics. This requirement ensures that the ordering defined by  $V_{\text{sim}}$  encodes feasible motions that solve the task in the real-world. This enables policy optimization algorithms to greedily follow  $V_{\text{sim}}$  at each step while ensuring that the resulting behavior successfully completes the task. We use the following pedagogical example to investigate why this is a reasonable property to assume for continuous control problems:

Pedagogical Example. Consider the following case where the real and simulated dynamics are both deterministic, namely  $s^0 = p_{\text{real}}(s; a)$  and  $s^0 = p_{\text{sim}}(s; a)$ . Specifically, consider the case where  $s = (s_1; s_2) \in \mathbb{S}^2$ ,  $a \in \mathbb{R}$ , and the dynamics are given by:

$$p_{\text{sim}}(s; a) = \begin{pmatrix} s_1 \\ s_2 \end{pmatrix} + t \begin{pmatrix} g \sin(s_1) \\ a \end{pmatrix}$$

$$p_{\text{real}}(s; a) = \begin{pmatrix} s_1 \\ s_2 \end{pmatrix} + t \begin{pmatrix} g \sin(s_1) \\ a + e(s_1; s_2) \end{pmatrix} :$$

These are the equations of motion for a simple pendulum (Wang et al., 2022) under an Euler discretization with time step  $t$ , where  $s_1$  is the angle of the arm,  $s_2$  is the angular velocity,  $a$  is the torque applied by the motor,  $g$  is the gravitational constant, and  $l$  is the length of the arm. The real-world dynamics contains unmodeled terms  $e(s_1; s_2)$ , which might correspond to complex frictional or damping effects. Consider the policy for the real world given by  $\pi_{\text{real}}(s) = \pi_{\text{sim}}(s) + e(s_1; s_2)$  and observe that  $\pi_{\text{sim}}(s; \pi_{\text{sim}}(s)) = p_{\text{real}}(s; \pi_{\text{real}}(s))$ . Even though the difference in transition dynamics  $e(s_1; s_2)$  might be quite large, the set of feasible next states is the same for the two environments. That is, the space of feasible motions in the two MDPs are identical, even though it takes substantially different policies to realize these motions

Geometric Insight. More broadly, if for each  $\epsilon$  there exists some  $\delta$  such that  $p_{\text{real}}(j; a) = p_{\text{sim}}(j; \pi_{\text{sim}}(s)) + \delta e(s_1; s_2)$ , then  $V_{\text{sim}}$  is improvable with respect to  $M_{\text{real}}$ . This follows from the fact that  $V_{\text{sim}}$  is improvable under  $p_{\text{sim}}$  by definition (see above discussion) and the fact that there exists actions which exactly match the state transitions in simulation and reality. More generally, it is reasonable to expect that  $\pi_{\text{sim}}$  approximately captures the geometry of what motions are feasible under  $p_{\text{real}}$ , even if the actions required to realize the motions differ substantially in the two MDPs. Thus it is reasonable to assume that  $V_{\text{sim}}$  is improvable. This intuition is highlighted by our real-world examples in cases where we use SFT with a prediction horizon of  $H = 1$ . In these cases the learned policy is able to greedily follow  $V_{\text{sim}}$  at each state and reach the goal, even in the face of large dynamics gaps. Relatedly, work on Lyapunov theory Westenbroek et al. (2022) has demonstrated theoretically that value functions are naturally robust to dynamics shifts under mild conditions.

**Main Theoretical Result.** Before presenting our main theoretical result, we provide a useful point of comparison from the literature Bhardwaj et al. (2020). When translated to our setting (Bhardwaj et al., 2020, Theorem 3.1) assumes (a) we have a) the value gap between simulation and reality is bounded by  $\|V_{sim}(s) - V_{real}(s)\| < \epsilon$  and b) that a generative model is used for planning where  $\|k(j; a) - p_{real}(j; a)\|_1 < \epsilon$ . Recalling that  $\hat{V}_H$  is the controller synthesized using the model  $p$ , (Bhardwaj et al., 2020, Theorem 3.1) bounds the suboptimality for the model-based controller as  $\|V_{real}(s) - V_{real}^{\hat{H}}(s)\| \leq \frac{\epsilon}{1-H} + \epsilon^H$ . To understand the bound, rst set  $\epsilon = 0$  so that there is no modeling error (as in the case of model-free instantiation SGFT). In this regime we are incentivized to increase  $H$  as this will decrease the  $\frac{\epsilon}{1-H}$  term and improve performance. However this term scales very poorly for long-horizon problems where  $H \rightarrow 1$ , and we may need large values of  $\epsilon$  to obtain near optimal policies. When  $\epsilon > 0$ , the situation becomes even more challenging. Increasing  $H$  will increase the  $\epsilon^H$  term, reflecting how longer prediction horizons propagate model-based errors and increase suboptimality. Thus, without additional structural assumptions, we cannot conclude SGFT can learn high-performance controllers when  $H$  is small. The following result demonstrates this is possible when  $V_{sim}$  is improvable:

**Theorem 1.** Assumes that a)  $\|V_{sim}(s) - V_{real}(s)\| < \epsilon$  and b) if a generative model  $p$  is used by SGFT then  $\|k(j; a) - p_{real}(j; a)\|_1 < \epsilon$ . Further suppose that  $V_{sim}$  is improvable with respect to  $M_{real}$ . Then for  $H$  sufficiently small and each  $s \in S$  we have:

$$\|V_{real}(s) - V_{real}^{\hat{H}}(s)\| \leq \frac{\epsilon}{1-H} + \epsilon^H; \tag{3}$$

where  $\hat{H}$  is the policy learned by SGFT.

See Appendix A for proof. This result demonstrates that when  $V_{sim}$  is improvable with respect to  $M_{real}$  SGFT can obtain nearly optimal policies, even when  $H$  is small. When compared to (Bhardwaj et al., 2020, Theorem 3.1), which assumes uniform worst-case bounds on  $V_{real}$ ,  $k$ , when  $V_{sim}$  is improvable we gain a substantial factor of  $\frac{1}{1-H}$  when bounding the effects of errors in  $V_{sim}$ . In short, this result demonstrates that even mild structural consistency between  $p_{real}$  and  $p_{sim}$  is enough to ensure that SGFT provides effective guidance towards performant policies when  $H$  is small. In the case of MBRL ( $\epsilon > 0$ ), this is especially important as keeping  $H$  small combats compounding errors in the dynamics model. Our proof technique adapts ideas from the theoretical control literature Grune & Rantzer (2008); Westebroek et al. (2022) to the more general setting we consider, and can be seen as a model-based analogy to the results from Cheng et al. (2021).

## 6 EXPERIMENTS

We answer the following (1) Can SGFT facilitate rapid online re-tuning for dynamic, contact-rich manipulation tasks? (2) Does SGFT improve the sample efficiency of re-tuning compared to baselines? (3) Can SGFT learn successful policies where prior methods fail entirely?

### 6.1 METHODS EVALUATED

**SGFT Instantiations.** We implement concrete instantiations of the general SGFT and MPC-SGFT frameworks sketched in Algorithms 2 and 3. SGFT-SAC is a model to real world transitions to perform data augmentation and uses SAC as a base model-free policy optimization algorithm. We use  $H = 1$  in all our experiments. Crucially, we set the ‘done’ flag to true at the end of each rollout – this ensures SAC does not bootstrap its own critic from the real-world data and only uses  $V_{sim}$  to bootstrap long-horizon returns. SGFT-TDMP-2 uses TDMP-2 Hansen et al. (2024) as a backbone. The base method learns a critic, a policy, and an approximate dynamics model through interactions with the environment. When acting in the world, the MPC controller solves online planning problems using the approximate model, the critic as a terminal reward, and uses the policy prior to seed an MPPI planner Williams et al. (2017). To integrate this method with SGFT, when transferring to the real world we simply freeze the critic learned in simulation and use the reshaped objective in Equation (1) as the online planning objective. For our experiments, we use  $H = 4$  and the default hyperparameters reported in Hansen et al. (2024).

<sup>1</sup>We note that Bhardwaj et al. (2020) focuses on general MPC problems where some approximation  $V_{real}$  is used as the terminal cost for the planning objective. However, despite the difference in settings, the structure of the underlying  $H$ -step objective we consider is identical to much of the MPC literature.

Figure 4: Real-world success rates during the course of online fine-tuning. We plot task success rates over number of fine-tuning rollouts for the tasks described in Sec. 6. We see that yields significant improvements in success and efficiency.

**Baseline Fine-tuning Methods.** We also fine-tune policies using the original in-domain horizon objective for  $M_{\text{real}}$ , including SAC (Haarnoja et al., 2018), TDMP2 (Hansen et al., 2024), QL (Kostrikov et al., 2021), PBR (fine-tunes the policy using the reshaped reward, an in-domain horizon, and SAC (Haarnoja et al., 2018)—it does not use horizon shorter RL), and RLPD (fine-tunes a fresh policy to solve the original MDP using RLPD (Ball et al., 2023)). These algorithms cover state-of-the-art model-based, model-free, and of-line-to-online adaptation algorithms.

**Baseline Sim-to-Real Methods.** Our Domain Randomization baseline refers to policies trained with extensive domain randomization in simulation and transferred directly to the real world. These policies rely only on the previous observations. Recurrent Policy + Domain Randomization uses policies conditioned on histories of observations, similar to methods such as Kumar et al. (2021). Asymmetric Actor-Critic (Pinto et al., 2018) uses policies trained in simulation with a critic conditioned on privileged information. DOREAMON (Tiboni et al., 2023b) is a recently proposed transfer method which automatically generates curricula to enable robust transfer. ASDr (Memmel et al., 2024) and DROPO (Tiboni et al., 2023a) identify simulation parameters with small real-world data sets. These methods serve as a state-of-the-art baselines for sim-to-real transfer methods.

## 6.2 SIM-TO-REAL EVALUATIONS

We test each method on five real-world manipulation tasks illustrated in Figure 3 and two additional real-world deformable object pushing tasks illustrated in Figure 1, demonstrating that both the SGFT-SAC and SGFT-TDMP2 instantiations of SGFT excel at learning policies with minimal real-world data.

Hammering is a highly dynamic task involving force and contact dynamics that are impractical to precisely model in simulation. In our setting, the robot is tasked with hammering a nail in a board. The nail has high, variable dry friction along its shaft. In order to hammer the nail into the board, the robot must hit the nail with high force repeatedly. The dynamics are inherently misspecified between simulation and reality here due to the infeasibility of accurately modeling the properties of the nail and its contact interaction with the hammer and board.

Figure 3: Sim-to-Real Setup. Simulation setup for pre-training (top) and execution of real-world fine-tuning (bottom) of real-world hammering (left), insertion force (middle), and pushing (right).

Insertion (Heo et al., 2023) involves the robot grasping a table leg and accurately inserting it into a table hole. The contact dynamics between the leg and the table differ between simulation and real-world conditions. In the simulation, the robot successfully completes the task by wiggling the

leg into the hole, but in the real world this precise motion becomes challenging due to inherent noise in the real-world observations as well as contact discrepancies between the leg and the table hole. Puck Pushing requires pushing a puck of unknown mass and friction forward to the edge of the table without it falling off the edge. Here, the underlying feedback controller of the real world robot inherently behaves differently from simulation. Additionally, retrieving and processing sensor information from cameras incurs variable amounts of latency. As a result, the controller executes each commanded action for variable amounts of time. These factors all contribute to the sim-to-real dynamics shift, requiring real-world re-tuning to reconcile.

Deformable Object Pushing is similar to puck pushing but involves pushing deformable objects which are challenging to model in simulation. As a result, we choose to model the object as a puck in simulation, making the simulation dynamics fundamentally misspecified compared to the real world. This example exemplifies how value functions trained under highly different dynamics in simulation can still provide useful behavioral priors for real-world exploration. We include two deformable pushing tasks – a towel and a squishy toy ball.

Each task is evaluated on a physical setup using a Franka FR3 operating with either Cartesian position control or joint position control at 5Hz. We compute object positions by color-thresholding pointclouds or by Aruco marker tracking, although this approach could easily be upgraded. Further details on reward functions, robot setups and environments can be found in Appendix B.

**Analysis.** The results of real-world re-tuning on these five tasks are presented in Figure 4. For all five tasks, zero-shot performance is quite poor due to the dynamics sim-to-real gap. The poor performance of direct sim-to-real transfer methods such as Domain Randomization, Recurrent Policy + Domain Randomization (Kumar et al., 2021), Asymmetric Actor-Critic (Pinto et al., 2018), ASID (Memmel et al., 2024), DROPO (Tiboni et al., 2023a), and DOREAMON (Tiboni et al., 2023b) highlight that these gaps are due to more than parameter misspecification or poor policy training in simulation, but rather stem from fundamental simulator misspecification.

The second class of comparison methods includes of-line pretraining with online re-tuning techniques like IQL (Kostrikov et al., 2021), SAC (Haarnoja et al., 2018), and RLPD (Ball et al., 2023). Whether model-free or model-based, the SGFT re-tuning methods (ours) substantially outperform these techniques in terms of efficiency and asymptotic performance. Moreover, they prevent catastrophic forgetting wherein re-tuning leads to periods of sharp degradation in the policies effectiveness. This suggests that simulation can offer more guidance during real-world policy search than just network weight initialization and/or replay buffer data initialization for subsequent re-tuning. Our full system consistently leads to significant improvement from re-tuning, achieving 100% success for hammering and pushing within an hour of re-tuning and 70% success for inserting within two hours of re-tuning. The fact that SGFT outperforms both TD-MPC2 (Hansen et al., 2024) and PBR-SAC, suggests that efficient re-tuning requires a combination of both short model rollouts and value-driven reward shaping. Last but not least, note that SGFT offers improvements on top of both SAC and TD-MPC2, showing the generality of the proposed paradigm.

In the Appendix we also include a set of sim-to-sim transfer benchmarks and hyper-parameter ablations (Appendix E) visualizations of transferred value functions (Appendix D).

## 7 LIMITATIONS AND FUTURE WORK

We present SGFT, a general framework for efficient sim-to-real re-tuning with existing RL algorithms. The key idea is to leverage value functions learned in simulation to provide guidance for real-world exploration. In the future it will be essential to scale methods to work directly from perceptual inputs. Calculating dense rewards from raw visual inputs is challenging, and represents an important limitation of the current instantiation of SGFT. Future work will investigate which visual modalities lead to robust sim-to-real transfer. Moreover, despite the sample complexity SGFT affords existing RL algorithms, we believe new base adaptation algorithms designed specifically for low-data regimes will be needed to make truly autonomous real world adaptation practical. For all methods tested, we found that tuning for real-world performance was time-consuming. Thus, future work will investigate using large simulated data sets to distill novel policy optimization algorithms that can be transferred to the real-world with less effort.

## 8 ACKNOWLEDGMENTS

We would like to thank Marius Memmel, Vidyaaranya Macha, Yunchu Zhang, Octi Zhang, and Arhan Jain for their invaluable engineering help. We would like to thank Lars Lien Ankile and Marcel Torne for their fruitful engineering discussions and constructive feedbacks. Patrick Yin is supported by sponsored research with Hitachi and the Amazon Science Hub.

## REFERENCES

- Brian Acosta, William Yang, and Michael Posa. Validating robotics simulators on real-world impacts. *IEEE Robotics and Automation Letters* 7(3):6471–6478, 2022.
- OpenAI: Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. Learning dexterous in-hand manipulation *The International Journal of Robotics Research* 39(1):3–20, 2020.
- Philip J Ball, Laura Smith, Ilya Kostrikov, and Sergey Levine. Efficient online reinforcement learning with offline data. In *International Conference on Machine Learning*, pp. 1577–1594. PMLR, 2023.
- Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemyslaw Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Christopher Hesse, Radekiewicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique Ponde de Oliveira Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang. Dota 2 with large scale deep reinforcement learning. *CoRR*, abs/1912.06680, 2019. URL <http://arxiv.org/abs/1912.06680>.
- Mohak Bhardwaj, Sanjiban Choudhury, and Byron Boots. Blending mpc & value function approximation for efficient reinforcement learning. *arXiv preprint arXiv:2012.05909*, 2020.
- Bernard Brogliato, Rogelio Lozano, Bernhard Maschke, Olav Egeland, et al. Dissipative systems analysis and control.
- Yevgen Chebotar, Ankur Handa, Viktor Makoviychuk, Miles Macklin, Jan Issac, Nathan D. Ratliff, and Dieter Fox. Closing the sim-to-real loop: Adapting simulation randomization with real world experience. In *ICRA*, 2019.
- Xinyue Chen, Che Wang, Zijian Zhou, and Keith Ross. Randomized ensembled double q-learning: Learning fast without a model. *arXiv preprint arXiv:2101.05982*, 2021.
- Zoey Chen, Aaron Walsman, Marius Memmel, Kaichun Mo, Alex Fang, Karthikeya Vemuri, Alan Wu, Dieter Fox, and Abhishek Gupta. Urdformer: A pipeline for constructing articulated simulation environments from real-world images. *arXiv preprint arXiv:2405.11656*, 2024.
- Ching-An Cheng, Xinyan Yan, Nathan Ratliff, and Byron Boots. Predictor-corrector policy optimization. In *International Conference on Machine Learning*, pp. 1151–1161. PMLR, 2019.
- Ching-An Cheng, Andrey Kolobov, and Adith Swaminathan. Heuristic-guided reinforcement learning. *Advances in Neural Information Processing Systems* 34:13550–13563, 2021.
- Open-X-Embodiment Collaboration. Open x-embodiment: Robotic learning datasets and rt-x models. In *ICRA*, 2024.
- Matt Deitke, Eli VanderBilt, Alvaro Herrasti, Luca Weihs, Kiana Ehsani, Jordi Salvador, Winson Han, Eric Kolve, Aniruddha Kembhavi, and Roozbeh Mottaghi. ProcTHOR: Large-scale embodied AI using procedural generation. *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022. URL [http://papers.nips.cc/paper\\_files/paper/2022/hash/27c546ab1e4f1d7d638e6a8dfbad9a07-Abstract-Conference.html](http://papers.nips.cc/paper_files/paper/2022/hash/27c546ab1e4f1d7d638e6a8dfbad9a07-Abstract-Conference.html).

- Yuqing Du, Olivia Watkins, Trevor Darrell, Pieter Abbeel, and Deepak Pathak. Auto-tuned sim-to-real transfer. In 2021 IEEE International Conference on Robotics and Automation (ICRA), pp. 1290–1296. IEEE, 2021.
- Frederik Ebert, Chelsea Finn, Sudeep Dasari, Annie Xie, Alex Lee, and Sergey Levine. Visual foresight: Model-based deep reinforcement learning for vision-based robotic control. arXiv preprint arXiv:1812.00568, 2018.
- Benjamin Eysenbach, Swapnil Asawa, Shreyas Chaudhari, Sergey Levine, and Ruslan Salakhutdinov. Off-dynamics reinforcement learning: Training for transfer with domain classification. arXiv preprint arXiv:2006.13916, 2020.
- Justin Fu, Avi Singh, Dibya Ghosh, Larry Yang, and Sergey Levine. Variational inverse control with events: A general framework for data-driven reward de-noising. In NeurIPS, 2018.
- Lars Grune and Anders Rantzer. On the infinite horizon performance of receding horizon controllers. IEEE Transactions on Automatic Control, 53(9):2100–2111, 2008.
- Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine. Continuous deep q-learning with model-based acceleration. In International conference on machine learning, pp. 2829–2838. PMLR, 2016.
- Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In ICLR, 2018. URL <http://proceedings.mlr.press/v80/haarnoja18b.html>.
- Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In International conference on machine learning, pp. 2555–2565. PMLR, 2019.
- Nicklas Hansen, Hao Su, and Xiaolong Wang. TD-MPC2: Scalable, robust world models for continuous control. In The Twelfth International Conference on Learning Representations, 2024. URL <https://openreview.net/forum?id=Oxh5CstDJU>.
- Minho Heo, Youngwoon Lee, Doohyun Lee, and Joseph J. Lim. Furniturebench: Reproducible real-world benchmark for long-horizon complex manipulation, 2023. URL <https://arxiv.org/abs/2305.12821>.
- Takuya Hiraoka, Takahisa Imagawa, Taisei Hashimoto, Takashi Onishi, and Yoshimasa Tsuruoka. Dropout q-functions for doubly efficient reinforcement learning. arXiv preprint arXiv:2110.02034, 2021.
- Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. Advances in neural information processing systems, 29, 2016.
- Hengyuan Hu, Suvir Mirchandani, and Dorsa Sadigh. Imitation bootstrapped reinforcement learning. arXiv preprint arXiv:2311.02198, 2023.
- Peide Huang, Xilun Zhang, Ziang Cao, Shiqi Liu, Mengdi Xu, Wenhao Ding, Jonathan Francis, Bingqing Chen, and Ding Zhao. What went wrong? closing the sim-to-real gap via differentiable causal discovery. In Jie Tan, Marc Toussaint, and Kouros Darvish (Eds.), Conference on Robot Learning, CoRL 2023, 6-9 November 2023, Atlanta, GA, USA, Volume 229 of Proceedings of Machine Learning Research, pp. 734–760. PMLR, 2023. URL <https://proceedings.mlr.press/v229/huang23c.html>.
- Ali Jadbabaie, Jie Yu, , and John Hauser. Unconstrained receding-horizon control of nonlinear systems. IEEE Transactions on Automatic Control, 46(5):776–783, 2001.
- Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. When to trust your model: Model-based policy optimization. Advances in neural information processing systems, 32, 2019.
- Rahul Kidambi, Aravind Rajeswaran, Praneeth Netrapalli, and Thorsten Joachims. Morel: Model-based off-line reinforcement learning. Advances in neural information processing systems, 33, 21810–21823, 2020.

- Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Of ine reinforcement learning with implicit q-learning. In *International Conference on Learning Representations*, 2021.
- Ashish Kumar, Zipeng Fu, Deepak Pathak, and Jitendra Malik. RMA: rapid motor adaptation for legged robots. In *RSS*, 2021.
- Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. Learning quadrupedal locomotion over challenging terrain. *Science robotics*, 5(47):eabc5986, 2020.
- Jacob Levy, Tyler Westenbroek, and David Fridovich-Keil. Learning to walk from three minutes of data with semi-structured dynamics models. In *14th Annual Conference on Robot Learning*, 2024. URL <https://openreview.net/forum?id=evCXwICMli>.
- Kevin Li, Abhishek Gupta, Ashwin Reddy, Vitchyr H Pong, Aurick Zhou, Justin Yu, and Sergey Levine. Mural: Meta-learning uncertainty-aware rewards for outcome-driven reinforcement learning. In *ICML*, 2021.
- Jinxin Liu, Hongyin Zhang, and Donglin Wang. Dara: Dynamics-aware reward augmentation in of ine reinforcement learning. *arXiv preprint arXiv:2203.06662*, 2022.
- Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Eureka: Human-level reward design via coding large language models. *arXiv preprint arXiv:2310.12931*, 2023.
- Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, and Gavriel State. Isaac gym: High performance gpu-based physics simulation for robot learning. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021.
- Ajay Mandlekar, Yuke Zhu, Animesh Garg, Jonathan Booher, Max Spero, Albert Tung, Julian Gao, John Emmons, Anchit Gupta, Emre Orbay, et al. Roboturk: A crowdsourcing platform for robotic skill learning through imitation. In *Conference on Robot Learning*, 2018.
- Gabriel B. Margolis, Ge Yang, Kartik Paigwar, Tao Chen, and Pulkit Agrawal. Rapid locomotion via reinforcement learning. *Int. J. Robotics Res.* 43(4):572–587, 2024. doi: 10.1177/02783649231224053. URL <https://doi.org/10.1177/02783649231224053>.
- Marius Memmel, Andrew Wagenmaker, Chuning Zhu, Patrick Yin, Dieter Fox, and Abhishek Gupta. ASID: active exploration for system identification in robotic manipulation. *CoRR abs/2404.12308*, 2024.
- Ashvin Nair, Abhishek Gupta, Murtaza Dalal, and Sergey Levine. Awac: Accelerating online reinforcement learning with of ine dataset. *arXiv preprint arXiv:2006.09359*, 2020.
- Mitsuhiko Nakamoto, Simon Zhai, Anikait Singh, Max Sobol Mark, Yi Ma, Chelsea Finn, Aviral Kumar, and Sergey Levine. Cal-ql: Calibrated of ine rl pre-training for efficient online netuning. *Advances in Neural Information Processing Systems*, 36, 2024.
- Andrew Y Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, volume 99, pp. 278–287, 1999.
- Haoyi Niu, Yiwen Qiu, Ming Li, Guyue Zhou, Jianming Hu, Xianyuan Zhan, et al. When to trust your simulator: Dynamics-aware hybrid of ine-and-online reinforcement learning. *Advances in Neural Information Processing Systems*, 35:36599–36612, 2022.
- Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE international conference on robotics and automation (ICRA)*, pp. 3803–3810. IEEE, 2018.
- Lerrel Pinto, Marcin Andrychowicz, Peter Welinder, Wojciech Zaremba, and Pieter Abbeel. Asymmetric actor critic for image-based robot learning. In *14th Robotics: Science and Systems, RSS 2018 MIT Press Journals*, 2018.

- Haozhi Qi, Ashish Kumar, Roberto Calandra, Yi Ma, and Jitendra Malik. In-hand object rotation via rapid motor adaptation. *ICoRL*, 2022. URL <https://proceedings.mlr.press/v205/qi23a.html>.
- Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, and Sergey Levine. Learning Complex Dexterous Manipulation with Deep Reinforcement Learning and Demonstrations. *Proceedings of Robotics: Science and Systems (RSS)* 2018.
- Fabio Ramos, Rafael Possas, and Dieter Fox. Bayessim: Adaptive domain randomization via probabilistic inference for robotics simulators. *RSS* 2019.
- Laura Smith, Ilya Kostrikov, and Sergey Levine. A walk in the park: Learning to walk in 20 minutes with model-free reinforcement learning. *arXiv preprint arXiv:2208.07860*, 2022a.
- Laura M. Smith, J. Chase Kew, Xue Bin Peng, Sehoon Ha, Jie Tan, and Sergey Levine. Legged robots that keep on learning: Fine-tuning locomotion policies in the real world. *ICRA*, 2022b.
- Wen Sun, J Andrew Bagnell, and Byron Boots. Truncated horizon policy search: Combining reinforcement learning & imitation learning. *arXiv preprint arXiv:1805.11240*, 2018.
- Richard S Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Machine learning proceedings 1990*, pp. 216–224. Elsevier, 1990.
- Richard S. Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *SIGART Bull.*, 2(4):160–163, jul 1991. ISSN 0163-5719. doi: 10.1145/122344.122377. <https://doi.org/10.1145/122344.122377>.
- DROID Collaboration team. Droid: A large-scale in-the-wild robot manipulation dataset. In *Robotics Science and Systems (RSS)* 2024.
- Gabriele Tiboni, Karol Arndt, and Ville Kyrki. Dropo: Sim-to-real transfer with of ine domain randomization. *Robotics and Autonomous Systems* 166:104432, 2023a.
- Gabriele Tiboni, Pascal Klink, Jan Peters, Tatiana Tommasi, Carlo D'Eramo, and Georgia Chaltatzaki. Domain randomization via entropy maximization. *arXiv preprint arXiv:2311.01885*, 2023b.
- Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *IROS* 2012.
- Marcel Torne, Anthony Simeonov, Zechu Li, April Chan, Tao Chen, Abhishek Gupta, and Pulkit Agrawal. Reconciling reality through simulation: A real-to-sim-to-real approach for robust manipulation. *CoRR* abs/2403.03949, 2024. doi: 10.48550/ARXIV.2403.03949. <https://doi.org/10.48550/arXiv.2403.03949>.
- Homer Walke, Kevin Black, Abraham Lee, Moo Jin Kim, Max Du, Chongyi Zheng, Tony Zhao, Philippe Hansen-Estruch, Quan Vuong, Andre He, Vivek Myers, Kuan Fang, Chelsea Finn, and Sergey Levine. Bridgedata v2: A dataset for robot learning at scale. *Conference on Robot Learning (CoRL)* 2023.
- Jin Wang, Qilong Xue, Lixin Li, Baolin Liu, Leilei Huang, and Yang Chen. Dynamic analysis of simple pendulum model under variable damping. *Alexandria Engineering Journal* 61(12): 10563–10575, 2022.
- Tingwu Wang and Jimmy Ba. Exploring model-based planning with policy networks. *arXiv preprint arXiv:1906.08649*, 2019.
- Tyler Westenbroek, Fernando Castaneda, Ayush Agrawal, Shankar Sastry, and Koushil Sreenath. Lyapunov design for robust and efficient robotic reinforcement learning. *arXiv preprint arXiv:2208.06721*, 2022.

- Grady Williams, Andrew Aldrich, and Evangelos A Theodorou. Model predictive path integral control: From theory to parallel computation. *Journal of Guidance, Control, and Dynamics*, 40(2):344–357, 2017.
- Maciej Wo czyk, Bart omiej Cupia , Mateusz Ostaszewski, Micha Bortkiewicz, Michal Zajkac, Razvan Pascanu, Lukasz Kucinski, and Piotr Milo's. Fine-tuning reinforcement learning models is secretly a forgetting mitigation problem. *arXiv preprint arXiv:2402.02868*, 2024.
- Kang Xu, Chenjia Bai, Xiaoteng Ma, Dong Wang, Bin Zhao, Zhen Wang, Xuelong Li, and Wei Li. Cross-domain policy adaptation via value-guided data iteration. *Advances in Neural Information Processing Systems*, 36:73395–73421, 2023.
- Tianhe Yu, Garrett Thomas, Lantao Yu, Stefano Ermon, James Y Zou, Sergey Levine, Chelsea Finn, and Tengyu Ma. Mopo: Model-based of ine policy optimization. *Advances in Neural Information Processing Systems*, 33:14129–14142, 2020.
- Wenhao Yu, Jie Tan, C. Karen Liu, and Greg Turk. Preparing for the unknown: Learning a universal policy with online system identification. *ICSS* 2017. URL <http://www.roboticsproceedings.org/rss13/p48.html>.
- Wenhao Yu, Nimrod Gileadi, Chuyuan Fu, Sean Kirmani, Kuang-Huei Lee, Montse Gonzalez Arenas, Hao-Tien Lewis Chiang, Tom Erez, Leonard Hasenclever, Jan Humplik, et al. Language to rewards for robotic skill synthesis. *arXiv preprint arXiv:2306.08647*, 2023.
- Marvin Zhang, Sharad Vikram, Laura Smith, Pieter Abbeel, Matthew Johnson, and Sergey Levine. Solar: Deep structured representations for model-based reinforcement learning. *International conference on machine learning*, pp. 7444–7453. PMLR, 2019.
- Yunchu Zhang, Liyiming Ke, Abhay Deshpande, Abhishek Gupta, and Siddhartha S. Srinivasa. Cherry-picking with reinforcement learning. *ICSS* 2023.
- Zhiyuan Zhou, Andy Peng, Qiyang Li, Sergey Levine, and Aviral Kumar. Ef cient online reinforcement learning ne-tuning need not retain of ine data. *arXiv preprint arXiv:2412.07762*, 2024.
- Brian D. Ziebart, Andrew Maas, J. Andrew Bagnell, and Anind K. Dey. Maximum entropy inverse reinforcement learning. *Proceedings of the 23rd National Conference on Arti cial Intelligence - Volume 3 AAAI'08*, pp. 1433–1438. AAAI Press, 2008. ISBN 9781577353683.

## A PROOFS

**Notation Recap.** We remind the reviewer of notation we have built up throughout the paper. We use the 'hat' notation to denote a generative dynamics model as well as the optimal values  $\hat{V}_H, \hat{Q}_H$  obtained by optimizing the H-step objective under these dynamics.  $\pi_H$  is then the policy obtained by optimizing the H-step objective under the model dynamics:  $\pi_H(s) = \arg \max_a \hat{Q}_H(s; a)$ .

We first present several Lemma's used in the proof of Theorem 1. The first result bounds the difference between the true H-step returns for a policy  $\pi_H$  and the H-step returns predicted under the dynamics model.

**Lemma 1.** (Bhardwaj et al., 2020, Lemma A.1.) Suppose  $\pi_H(s; a) = \arg \max_a p_{\text{real}}(s; a)k_1$ . Further suppose  $r = \max_s r(s) - \min_s r(s)$  and  $V = \max_s V_s(s) - \min_s V_s(s)$  are finite. Then, for each policy  $\pi_H$  we may bound the H-step returns under the model and true dynamics by:

$$k_1 \hat{V}_H^{\pi_H}(s) - V_H^{\pi_H}(s) \leq \frac{1}{1 - \gamma} \frac{r}{2} + \gamma \frac{V}{2} \quad \forall H \geq 1 \quad (4)$$

**Proof.** This result follows immediately from the proof of (Bhardwaj et al., 2020, Lemma A.1.), with changes to notation and noting that we assume access to the true reward. In particular, the full result of (Bhardwaj et al., 2020, Lemma A.1.) includes an extra  $\gamma^H$  term which comes from the usage of a model  $\hat{r}$  which estimates the true reward. We do not consider such effects, and thus suppress this dependence.  $\square$

The next result uses this 1 to bound the difference between the optimal H-step returns and the H-step returns generated by the policy  $\pi_H$  which is optimal under the dynamics model.

**Lemma 2.** Suppose that  $\pi_H(s; a) = \arg \max_a p_{\text{real}}(s; a)k_1$ . Further suppose  $r = \max_s r(s) - \min_s r(s)$  and  $V = \max_s V_s(s) - \min_s V_s(s)$  are finite. Then for each state  $s \in \mathcal{S}$  we have:

$$V_H(s) - V_H^{\pi_H}(s) \leq \frac{1}{1 - \gamma} \frac{r}{2} + \gamma \frac{V}{2} \quad (5)$$

where  $V_H^{\pi_H} = \max_{\pi_H} \hat{V}_H^{\pi_H}(s)$ .

**Proof.** Let  $\pi_H^* = \arg \max_{\pi_H} V_H^{\pi_H}(s)$  be the optimal H-step policy under the true dynamics. By Lemma 1 we have both that

$$V_H(s) - \hat{V}_H^{\pi_H^*}(s) \leq \frac{1}{1 - \gamma} \frac{r}{2} + \gamma \frac{V}{2} \quad \forall H \geq 1 \quad (6)$$

$$\hat{V}_H^{\pi_H}(s) - V_H^{\pi_H}(s) \leq \frac{1}{1 - \gamma} \frac{r}{2} + \gamma \frac{V}{2} \quad \forall H \geq 1 \quad (7)$$

Combining these two bounds with the fact that  $V_H(s) = \hat{V}_H^{\pi_H^*}(s)$  yields the desired result.  $\square$

The following result establishes an important monotonicity property on the optimal H-step value functions which is important for the main result.:

**Lemma 3.** Suppose that  $\sup_a E_{s \sim p_{\text{real}}(s; a)} [V_{\text{sim}}(s^0)] - V_{\text{sim}}(s) > r(s)$ . Then we have  $V_H(s) > V_{H-1}(s)$  for each  $s \in \mathcal{S}$ . Then for each  $s \in \mathcal{S}$  we have:

$$V_H(s) > V_{H-1}(s) \quad (8)$$

**Proof.** Fix an initial condition  $s_0 \in \mathcal{S}$ . Let  $\pi_H$  be arbitrary, and let  $\pi_H^*$  be the shorthand  $\pi_H^* = \arg \max_{\pi_H} V_H^{\pi_H}(s_0)$  for the time-varying policy  $\pi_H^* = \arg \max_{\pi_H} V_H^{\pi_H}(s_0)$ . Then, concatenate these policies to define  $\pi_H = \pi_H^* \circ \pi_H$ , which is simply the result of applying the optimal policy for the (H-1)-step look ahead objective Equation (1) starting from  $s_0$  followed by applying  $\pi_H^*$  for a single step. Letting the following distributions over trajectories be generated by the definition

of  $V_H$ :

$$\begin{aligned}
 & V_H(s_0) \\
 & \leq E^H [V_{\text{sim}}(s_H) + \sum_{t=1}^{H-1} \gamma^t r(s_t) + \gamma^H V_{\text{sim}}(s_0)] \\
 & = E^H [V_{\text{sim}}(s_H) + \sum_{t=1}^{H-1} \gamma^t V_{\text{sim}}(s_{H-t}) + \sum_{t=1}^{H-1} \gamma^t r(s_{H-t}) + \gamma^H V_{\text{sim}}(s_0)] \\
 & = E^H [V_{\text{sim}}(s_H) + \sum_{t=1}^{H-1} \gamma^t V_{\text{sim}}(s_{H-t}) + \sum_{t=1}^{H-1} \gamma^t r(s_{H-t}) + V_{H-1}(s_0)]
 \end{aligned}$$

The first inequality follows from the fact that the return of cannot be greater than that of, the first equality follows from rearranging terms to isolate  $V_{H-1}$ , and the second equality follows from the definition of  $V_{H-1}$ . Now, since our choice of used to define was arbitrary, we choose to be deterministic and such that  $V_{\text{sim}}(s) > r(s)$  at each state  $s$ , as guaranteed by the assumption made for the result. This choice of policy grants that:

$$E^H [V_{\text{sim}}(s_H) - \sum_{t=1}^{H-1} \gamma^t V_{\text{sim}}(s_{H-t}) + \sum_{t=1}^{H-1} \gamma^t r(s_{H-t})] > 0 \quad (9)$$

The desired result follows immediately by combining the two preceding bounds, and noting that our choice of initial condition was arbitrary, meaning the preceding analysis holds for all initial conditions.  $\square$

Our final lemma bounds the sub-optimality of GFT policies in terms of a) errors in the sim value function and b) additional suboptimality caused by being sub-optimal for the  $H$ -step objective:

Lemma 4. Suppose that  $V_{\text{sim}}$  is improvable and further suppose that  $\max_{s \in \mathcal{S}} |V_{\text{sim}}(s) - V_{\text{real}}(s)| < \epsilon$ . Then any policy which satisfies  $A_H(s; \pi) = Q_H(s; \pi) - V_H(s)$  will satisfy:

$$V_{\text{real}}(s) - V_{\text{real}}(s) \leq \epsilon + \frac{\epsilon}{1-\gamma} \quad (10)$$

Proof. Our goal is first to bound how  $Q_H(s; \pi)$  changes on expectation when applying the given policy for a single step. We have that:

$$Q_H(s; \pi) + \gamma V_H(s) \quad (11)$$

$$V_H(s) - V_{H-1}(s) \quad (12)$$

$$Q_H(s; \pi) = E[V_{H-1}(s^0) + r(s; s^0)] \quad (13)$$

where the first inequality follows from the assumption of the theorem, the second inequality follows from Lemma 3 and, the third inequality is simply the definition of  $Q_H$ . Letting  $s^0 \sim p_{\text{real}}(s; a)$  with  $a = \pi(s)$ , we can take expectations can combine the previous relations to obtain:

$$E[Q_H(s^0; \pi) + r(s; s^0)] + \gamma E[V_H(s^0) + r(s; s^0)] - E[V_{H-1}(s^0) + r(s; s^0)] = Q_H(s; \pi) \quad (14)$$

That is:

$$E[Q_H(s^0; \pi)] + r(s) + \gamma Q_H(s; \pi) \quad (15)$$

Alternatively:

$$r(s) - Q_H(s; \pi) \leq E[Q_H(s^0; \pi)] - r(s) \quad (16)$$

Next, we use this bound to provide a lower bound  $V_{\text{real}}(s)$ . Because the previous analysis holds at all states when we apply, the following holds over the distribution of trajectories generated by

applying starting from the initial conditions  $s_0$ :

$$\begin{aligned} E_{\text{real}}(s) &= \sum_{t=0}^{\infty} \gamma^t r(s_t) = V_{\text{real}}(s) - V_{\text{real}}(s_0) \\ &= \sum_{t=0}^{\infty} \gamma^t (Q_H(s_t; \pi) - Q_H(s_{t+1}; \pi)) \\ &= Q_H(s_0; \pi) - \lim_{t \rightarrow \infty} \gamma^t Q_H(s_t; \pi); \end{aligned}$$

where we have repeatedly telescoped out sums to cancel out terms in the first equality, used (16) in the second equality, and canceled out terms to generate the final equality.

Thus, we have the lower-bound:

$$\begin{aligned} V_{\text{real}}(s) &\geq Q_H(s; \pi) + V_{\text{sim}}(s_0) - \lim_{t \rightarrow \infty} \gamma^t Q_H(s_t; \pi) \\ &= V_H(s) + V_{\text{sim}}(s_0) - \lim_{t \rightarrow \infty} \gamma^t Q_H(s_t; \pi); \end{aligned} \quad (17)$$

Next, we may bound:

$$\begin{aligned} V_H(s_0) + V_{\text{sim}}(s_0) &\leq E_{\text{real}}(s_0) + \sum_{t=0}^{\infty} \gamma^t r(s_t) \\ &= E_{\text{real}}(s_0) + \sum_{t=0}^{\infty} \gamma^t (Q_H(s_t; \pi) - Q_H(s_{t+1}; \pi)) \\ &= E_{\text{real}}(s_0) + Q_H(s_0; \pi) - \lim_{t \rightarrow \infty} \gamma^t Q_H(s_t; \pi); \end{aligned} \quad (18)$$

Invoking the assumption that  $\max_s |V_{\text{sim}}(s) - V_{\text{real}}(s)| < \epsilon$ , we can combine this with the preceding bound to yield:

$$V_H(s_0) + V_{\text{sim}}(s_0) - V_{\text{real}}(s_0) \leq \epsilon. \quad (19)$$

Finally, once more invoking the fact that  $Q_H(s; \pi) + V_H(s)$  for each  $s \in \mathcal{S}$  and combining this with Equation (17) and Equation (18), we obtain that:

$$\begin{aligned} V_{\text{real}}(s) &\geq Q_H(s; \pi) + V_{\text{sim}}(s_0) - \lim_{t \rightarrow \infty} \gamma^t Q_H(s_t; \pi) \\ &\geq V_H(s) + V_{\text{sim}}(s_0) - \lim_{t \rightarrow \infty} \gamma^t Q_H(s_t; \pi) \\ &\geq V_{\text{real}}(s) - \epsilon + V_{\text{sim}}(s_0) - \lim_{t \rightarrow \infty} \gamma^t Q_H(s_t; \pi) \\ &= V_{\text{real}}(s) - \epsilon + \lim_{t \rightarrow \infty} \gamma^t Q_H(s_t; \pi) - \lim_{t \rightarrow \infty} \gamma^t Q_H(s_t; \pi) \\ &= V_{\text{real}}(s) - \epsilon. \end{aligned}$$

from which the state result follows immediately.  $\square$

Proof of Theorem 1:

Proof. The result follows directly from a combination of Lemma 4 and Lemma 2 by suppressing problem-dependent constants and lower order terms in the discount factor  $\gamma$ .  $\square$

## B ENVIRONMENT DETAILS

**Sim2Real Environment.** We use a 7-DoF Franka FR3 robot with a 1-DoF parallel-jaw gripper. Two calibrated Intel Realsense D455 cameras are mounted across from the robot to capture position of the object by color-thresholding pointcloud readings or retrieving pose estimation from aruco tags. Commands are sent to the controller at 5Hz. We restrict the end-effector workspace of the robot in a rectangle for safety so the robot arm doesn't collide dangerously with the table and objects outside

the workspace. We conduct extensive domain randomization and randomize the initial gripper pose during simulation training. The reward is computed from measured proprioception of the robot and estimated pose of the object. Details for each task are listed below.

**Hammering.** For hammering, the action is 3-dimensional and sets delta joint targets for 3 joints of the robot using joint position control. The observation space is 12-dimensional and includes end-effector cartesian xyz, joint angles of the 3 movable joints, joint velocities of the 3 movable joints, the z position of the nail, and the xz position of the goal. Each trajectory is 50 timesteps. In simulation, we randomize over the position, damping, height, radius, mass, and thickness of the nail. Details are listed in Tab. 1.

The reward function is parameterized as  $r(t) = 10 \cdot r_{\text{nail\_goal}}(t)$  where  $r_{\text{nail\_goal}} = (r_{\text{nail}})_z - (r_{\text{goal}})_z$  represents the distance in the z dimension of the nail head to the goal, which we set to be the height of the board the nail is on.

**Pushing.** For pushing, the action is 2-dimensional and sets delta cartesian xy position targets using end-effector position control. The observation space is 4-dimensional and includes end-effector cartesian xy and the xy position of the puck object. Each trajectory is 40 timesteps. In simulation, we randomize over the position of the puck. Details are listed in Tab. 3.

Let  $r_{ee}$  be the cartesian position of the end effector and  $r_{obj}$  be the cartesian position of the object. The reward function is parameterized as  $r(t) = r_{ee\_goal}(t) - r_{obj\_goal}(t) + r_{\text{threshold}}(t) - r_{\text{table}}(t)$  where  $r_{ee\_goal}(t) = k(r_{ee}(t) - r_{obj}(t) + [3.5\text{cm}; 0.0\text{cm}; 0.0\text{cm}])$  represents the distance of the end effector to the back of the puck,  $r_{obj\_goal}(t) = k(r_{obj}(t))_x - 55\text{cm}$  represents the distance of the puck to the goal (which is the edge of the table along the x dimension),  $r_{\text{threshold}}(t) = I[r_{obj\_goal}(t) > 2.5\text{cm}]$  represents a goal reaching binary signal, and  $r_{\text{table}}(t) = I[(r_{obj}(t))_z > 0.0]$  represents a binary signal for when the object falls of the table.

**Inserting.** For inserting, the action is 3-dimensional and sets delta cartesian xyz position targets using end effector position control. The observation space is 9-dimensional and includes end-effector cartesian xyz, the xyz of the leg, and the xyz of the table hole. Each trajectory is 40 timesteps. We enlarge the table by a scale of 1.08 compared in the original table in FurnitureBench in order to make the table leg insertable without twisting. In simulation, we randomize the initial gripper position, position of the table, and friction of both the table and the leg.

Let  $r_{\text{pos1}}(t)$  and  $r_{\text{pos2}}(t)$  represent the Cartesian positions of the leg and table hole. Let:

$$\begin{aligned} x_{\text{distance}}(t) &= \text{clip}(|r_{\text{pos1x}}(t) - r_{\text{pos2x}}(t)|; 0.0; 0.1) \\ y_{\text{distance}}(t) &= \text{clip}(|r_{\text{pos1y}}(t) - r_{\text{pos2y}}(t)|; 0.0; 0.1) \\ z_{\text{distance}}(t) &= \text{clip}(|r_{\text{pos1z}}(t) - r_{\text{pos2z}}(t)|; 0.0; 0.1) \end{aligned}$$

Let the success condition be defined as:

$$r_{\text{success}}(t) = I[x_{\text{distance}}(t) < 0.01 \text{ and } y_{\text{distance}}(t) < 0.01 \text{ and } z_{\text{distance}}(t) < 0.01]$$

The reward function is now:

$$r(t) = r_{\text{success}}(t) - 100 \cdot (x_{\text{distance}}(t)^2 + y_{\text{distance}}(t)^2 + z_{\text{distance}}(t)^2)$$

**Sim2Sim Environment.** We additionally attempt to model a sim2real dynamics gap in simulation by taking the hammering environment and create a proxy for the real environment by fixing the domain randomization parameters, fixing the initial gripper pose, and rescaling the action magnitudes before rolling out in the environment.

## C IMPLEMENTATION DETAILS

**Algorithm Details.** We use SAC as our base off-policy RL algorithm for training in simulation and fine-tuning in the real world. For our method, we additionally add in two networks: a dynamics model that predicts next state given current state and action, and a state-conditioned value network which regresses towards the Q-value estimates for actions taken by the current policy. These networks are training jointly with the actor and critic during SAC training in simulation.

**Network Architectures.** The Q-network, value network, and dynamics model are all parameterized by a two-layer MLP of size 512. The dynamics model is implemented as a delta dynamics model where model predictions are added to the input state to generate next states. The policy network produces the mean  $\mu_a$  and a state-dependent log standard deviation  $\log \sigma_a$  which is jointly

Table 1: Domain randomization of hammering task in simulation

| Name                    | Range           |
|-------------------------|-----------------|
| Nail x position (m)     | [0.3, 0.4]      |
| Nail z position (m)     | [0.55, 0.65]    |
| Nail damping            | [250.0, 2500.0] |
| Nail half height (m)    | [0.02, 0.06]    |
| Nail radius (m)         | [0.005, 0.015]  |
| Nail head radius (m)    | [0.03, 0.04]    |
| Nail head thickness (m) | [0.001, 0.01]   |
| Hammer mass (kg)        | [0.015, 0.15]   |

Table 2: Domain randomization of pushing task in simulation

| Name                  | Range         |
|-----------------------|---------------|
| Object x position (m) | [0.0, 0.3]    |
| Object y position (m) | [-0.25, 0.25] |

Table 3: Domain randomization of inserting task in simulation

| Name                     | Range         |
|--------------------------|---------------|
| Parts x/y position (m)   | [-0.05, 0.05] |
| Parts rotation (degrees) | [0, 15]       |
| Parts friction           | [-0.01, 0.01] |

learned from the action distribution. The policy network is parameterized by a two-layer MLP of size 512, with a mean head and log standard deviation head on top parameterized by a FC layer.

**Pretraining in Simulation.** For hammering and puck pushing, we collect 25,000,000 transitions of random actions and pre-compute the mean and standard deviation of each observation across this dataset. We train SAC in simulation on the desired task by sampling 50-50 from the random action dataset and the replay buffer. We normalize our observations by the pre-computed mean and standard deviation before passing them into the networks. We additionally add Gaussian noise centered at 0 with standard deviation 0.004 to our observations with 30% probability during training. For inserting, we train SAC in simulation with no normalization. We train SAC with autotuned temperature set initially to 1 and a UTD of 1. We use Adam optimizer with a learning rate of  $3 \cdot 10^{-4}$ , batch size of 256, and discount factor  $\gamma = 0.99$ .

**Fine-tuning in Real World.** We pre-collect 20 real-world trajectories with the policy learned in simulation to fill the empty replay buffer. We then reset the critic with random weights and continue training SAC with a fixed temperature of  $\tau = 0.01$  and with a UTD of 2 with the pretrained actor and dynamics model. We freeze the value network learned from simulation and use it to relabel PBRs rewards during fine-tuning. During fine-tuning, for each state sampled from the replay buffer, we additionally hallucinate 5 branches off and add it to the training batch. As a result, our batch size effectively becomes 1536. The policy, Q-network, and dynamics model are all trained jointly on the real data during SAC fine-tuning. We don't train on any simulation data during real-world fine-tuning because we empirically found it didn't help fine-tuning performance in our settings.

Figure 5: Visualization of real rollout, hallucinated states, and value function. The red dots indicate states along a real rollout in simulation. The blue dots indicate hallucinated states branching off real states generated by the learned dynamics model. The green heatmap indicates the value function estimates at different states. A corresponding image of the state is shown for two states. Since it is hard to directly visualize states and values due to the high-dimensionality of the state space, we only show a part of the trajectory where the puck does not move. This allows us to visualize states and values along changes in only end effector  $xy$ .

## D QUALITATIVE RESULTS

We analyze the characteristics of hallucinated states and value functions in Fig. 5. We visualize a trajectory of executing puck pushing in simulation using the learned policy in this plot. The red dots indicate states along a real rollout in simulation. The blue dots indicate hallucinated states branching off real states generated by the learned dynamics model. The green heatmap indicates the value function estimates at different states. A corresponding image of the state is shown for two states. The trajectory shown in the figure shows the learned policy moving closer to the puck before pushing it. The value function heatmap shows higher values when the end effector is closer to the puck and lower values when further. Hallucinated states branching off each state show generated states for re-tuning the learned policy.

Note that it is hard to directly visualize states and values due to the high-dimensionality of the state space. To get around this for puck pushing, we only show a part of the trajectory where the puck does not move. This allows us to visualize states and values along changes in only end effector  $xy$ .

