Provably sample-efficient RL with side information about latent dynamics

Yao Liu*
Amazon Web Services
yaoliuai@amazon.com

Dipendra Misra
Microsoft Research
dipendra.misra@microsoft.com

Miroslav Dudík
Microsoft Research
mdudik@microsoft.com

Robert E. Schapire
Microsoft Research
schapire@microsoft.com

Abstract

We study reinforcement learning (RL) in settings where observations are high-dimensional, but where an RL agent has access to abstract knowledge about the structure of the state space, as is the case, for example, when a robot is tasked to go to a specific room in a building using observations from its camera, while having access to the floor plan. We formalize this setting as transfer reinforcement learning from an *abstract simulator*, which we assume is deterministic (such as a simple model of moving around the floor plan), but which is only required to capture the target domain's latent-state dynamics approximately up to *unknown* (bounded) perturbations (to account for environment stochasticity). Crucially, we assume no prior knowledge about the structure of observations in the target domain except that they can be used to identify the latent states (but the decoding map is unknown). Under these assumptions, we present an algorithm, called TASID, that learns a robust policy in the target domain, with sample complexity that is polynomial in the horizon, and *independent* of the number of states, which is not possible without access to some prior knowledge. In synthetic experiments, we verify various properties of our algorithm and compare it with several transfer RL algorithms that require access to "full simulators" (i.e., those that also simulate observations).

1 Introduction

When learning from scratch, reinforcement learning (RL) in the real world can be very expensive. For example, a robot learning to navigate in a building might need to explore every possible state or location, which can be painstakingly costly and time-consuming. Sometimes, however, it is not necessary to begin such a learning process from scratch. For instance, in the robot example, we might have access to a general floor map of the building. How can this kind of high-level but imprecise information be used to learn how to operate in the environment more quickly and more effectively?

In this paper, we study how to effectively leverage prior information in the form of such "abstract" descriptions of the environment. We formalize this abstract description as an "abstract simulator," which, like a map, can be used as an imperfect model. Importantly, our abstract simulators differ from more standard simulators in that they only focus on the "latent structure" of the environment dynamics, not on the observations that might be experienced by an agent in the environment.

In general, fully faithful simulators of even the latent dynamics might be hard to build, for instance, due to the difficulty of exactly modeling all probabilistic outcomes, as when the robot's actions do

*The work was done while the author was an intern at Microsoft Research and a graduate student at Stanford.

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

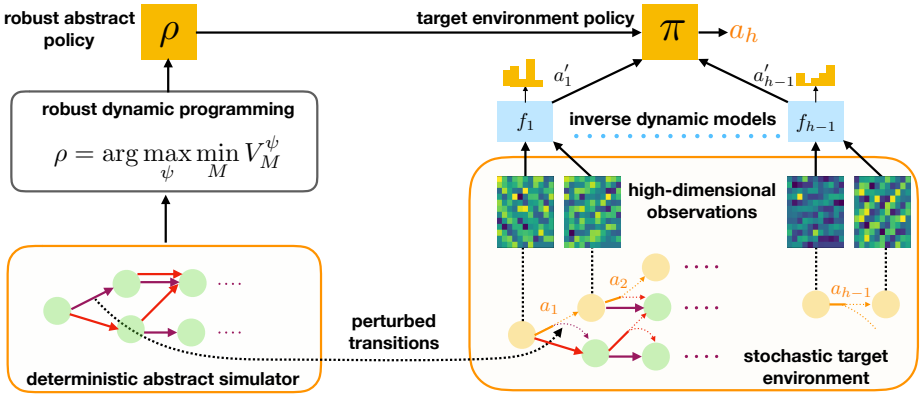


Figure 1: Overview of our setup. We propose an algorithm TASID that learns a robust policy π in the target environment using side information in the form of a fully specified deterministic model, called an "abstract simulator." Abstract simulator approximates (possibly stochastic) latent dynamics of the target environment and thus serves as an idealized description of the the environment. The policy π selects its actions according to the robust abstract policy ρ, but the latent states need to be inferred from high-dimensional observations using learned inverse dynamics models.

not have exactly their intended effect. More complex simulators might be more faithful, but simpler simulators might be easier to build and also more computationally tractable.

In this paper, we address this trade-off with a particular design choice: First, we assume that the abstract simulator is deterministic. Indeed, an ordinary map, which implicitly represents what new position will be reached by a particular action, is such a deterministic model. Compared to fully probabilistic models, deterministic ones are especially simple to build and work with.

On the other hand, a deterministic model will not in general capture the "noise" of real-world dynamics. Therefore, we seek algorithms that will learn policies that are robust to errors or "perturbations" in the model represented by the abstract simulator. To this end, in Section 4, we present a new algorithm called TASID that provably finds a policy achieving a certain level of performance for any target environment whose dynamics can be reasonably approximated by a given abstract simulator, up to some quantifiable perturbation level, and with arbitrarily different observations. (Figure 1 visualizes our setup and approach.)

In the robot example, although a map is intuitively helpful for reducing the need for exploration, there is still much to be inferred, for instance, how locations on the map correspond to locations in the building, and, even more challenging, how they correspond to rich, high-dimensional observations experienced via cameras or other sensors. Our algorithm solves these challenges, building on prior work on learning from such observations [Krishnamurthy et al., 2016, Jiang et al., 2017].

Furthermore, our theoretical guarantees show that we do indeed save dramatically on sample complexity (measured by the required number of interactions with the target environment) by not having to explore the entire environment, and instead focusing the task of learning observations just to the states that lead towards accomplishing the task. Indeed, our algorithm's sample complexity is entirely *independent* of the size of the state space, assuring efficiency even when the state space is extremely large, as is often the case. We are able to achieve this result because of specific assumptions and criteria for success, as outlined above, namely, near-deterministic latent dynamics, knowledge of the abstract simulator, and the benchmark of a robust policy rather than an optimal policy.

In Section 5, we empirically evaluate TASID in two domains. The first represents a challenging problem requiring strategic exploration. We show that TASID is able to efficiently achieve the robust policy value while the PPO algorithm [Schulman et al., 2017] augmented with exploration bonus [Burda et al., 2019b] and domain randomization [Tobin et al., 2017] does not solve the problem. The second domain is a visual grid world that tests the robustness of TASID to dynamics perturbation and the change of state-space size. We show that TASID succeeds empirically in agreement with our theory.

2 Related Work

Transfer RL. The goal of transfer RL is to speed up learning in a target environment by leveraging a model (or some other form of knowledge) learnt in a source environment [Taylor and Stone, 2009, Lazaric, 2012]. For our setting, the most relevant are the methods that can transfer between environments with different observation spaces. The dominant line of research requires access to a hand-coded inter-task mapping, which is used to introduce constraints, shape rewards, or warm-start learning in the target environment [Taylor and Stone, 2005, Taylor et al., 2007a, Fernández et al., 2010, Mann and Choe, 2013, Brys et al., 2015]. Instead of hand-coding, several works seek to learn inter-task mappings using structural information such as factorization [Liu and Stone, 2006, Taylor et al., 2007b, Soni and Singh, 2006]. Van Driessel and Francois-Lavet [2021] and Sun et al. [2022] do not require such structural information, and only posit assumptions similar to our block MDP assumption, but without requiring near-deterministic latent dynamics. Latent dynamics learnt in the source domain is used to create auxiliary tasks that regularize learning in the target domain.

These methods show empirical promise of transferring information about latent dynamics. Their focus is mainly on modeling and experiments, whereas our work also seeks to derive theoretical guarantees that showcase the benefits of transfer. We derive sample complexity that is independent of the size of the state space and observation space, which would be impossible without transfer. Among the cited works, Mann and Choe [2013] also show benefits of transfer, but their sample complexity generally scales linearly with the size of the target *observation* space. The strong bound that we derive requires a near-deterministic latent dynamics, and so it is not as broadly applicable.

Provably efficient RL with rich observations. Our modeling setup and theoretical analysis build on the prior work that considers large-dimensional "rich" observations, but simpler (tabular) latent dynamics [Krishnamurthy et al., 2016, Jiang et al., 2017, Du et al., 2019]. To circumvent the hardness of learning in general partially-observed MDPs (POMDPs), these methods impose structural assumptions, such as the block MDP assumption used here. These methods, like ours, learn with sample complexity that is independent of the size of the observation space, but, unlike ours, that is polynomial in the size of the latent state space. Using existing MDP lower bounds [Kakade, 2003], this dependence is unavoidable for general latent dynamics, even when the decoding function of observations to latent states is known. We overcome this fundamental obstacle by assuming that the latent dynamics is near-deterministic, and the deterministic approximation is known.

Robust RL. Our solution concept and the underlying algorithm build on robust dynamic programming methods [Iyengar, 2005, Bagnell et al., 2001]. However, we are using this concept in a fundamentally new way to achieve sample complexity improvements in transfer learning. This is one of the key conceptual innovations of our work.

3 Problem Setting

We next formalize our modeling setup and assumptions. As a running example, we consider a navigation problem on a floor of a building, where the goal is to reach a certain location by a robot that can turn left and right and move forward. The robot senses the environment via lidar readings and/or a camera, but it does not have access to its position or orientation. We use the notation $\Delta(S)$ for the set of probability distributions over a set S, and $[n]$ to denote the set $\{1, 2, \ldots, n\}$.

3.1 Target environment

We assume that the target environment is a *block MDP* (see, e.g., Du et al., 2019, Misra et al., 2020, Zhang et al., 2020). In a block MDP, observations are high-dimensional (e.g., lidar and camera readings), but emitted by a finite number of latent states (e.g., location and orientation of a robot); latent states are not directly observed, but they can be determined from the observations they emit.

Formally, a block MDP is a triple $\mathbf{M} = \langle M, \mathcal{X}, q \rangle$. The first component of the triple is a standard episodic MDP $M = \langle \mathcal{S}, \mathcal{A}, s_{\text{init}}, H, T, R \rangle$, referred to as the *latent MDP*, with a finite state space $\mathcal{S}$, referred to as the *latent state space*, a finite action space $\mathcal{A}$, initial latent state s_{init}, horizon H, transition function $T : [H] \times \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$, and reward function $R : [H] \times \mathcal{S} \times \mathcal{A} \to [0, 1]$; transition probabilities and reward function values are written as $T_h(s' \mid s, a)$ and $R_h(s, a)$. The second component of the block MDP triple is an *observation space* $\mathcal{X}$, which is typically large and

possibly infinite. The final component is an *emission function* $q : \mathcal{S} \to \Delta(\mathcal{X})$, which describes the conditional distribution over observations given any latent state; its values are written as $q(s \mid x)$.

We assume that q satisfies the block MDP assumption, meaning that there exist disjoint sets $\{\mathcal{X}_s\}_{s\in\mathcal{S}}$ such that if $x \sim q(\cdot \mid s)$ then $x \in \mathcal{X}_s$ with probability 1. This means that there exists a *perfect decoder* $\phi_{\mathbf{M}} : \mathcal{X} \to \mathcal{S}$ that maps an observation x to the unique state $\phi_{\mathbf{M}}(x)$ that emits it.

An agent interacts with a block MDP in a sequence of episodes, each generated as follows: Initially, $s_1 = s_{\text{init}}$. At each step $h = 1, \ldots, H$, the agent observes $x_h \sim q(\cdot \mid s_h)$, then takes action a_h, accrues reward $r_h = R_h(s_h, a_h)$, after which the MDP transitions to state $s_{h+1} \sim T_h(\cdot \mid s_h, a_h)$. The agent does *not* observe the latent states s_h, only the observations x_h and rewards r_h. Observations and actions up to step h are denoted $\mathbf{x}_{1:h}$ and $\mathbf{a}_{1:h}$.

We denote the block MDP describing the target environment by $\mathbf{M}^\star = \langle M^\star, \mathcal{X}, q^\star\rangle$ where $M^\star = \langle \mathcal{S}, \mathcal{A}, s_{\text{init}}, H, T^\star, R^\star\rangle$ is the target latent MDP. The perfect decoder for $\mathbf{M}^\star$ is denoted as $\phi^\star$.

Practicable policy. Behavior of an agent in a target environment is formalized as a (non-Markovian) *practicable policy*. A practicable policy prescribes which action to take given any sequence of observations, i.e., it is a mapping $\pi : \mathcal{X}^{\leq H} \to \mathcal{A}$, where $\mathcal{X}^{\leq H} = \cup_{h=1}^{H} \mathcal{X}^h$. The action taken by π on $\mathbf{x}_{1:h}$ is written $\pi_h(\mathbf{x}_{1:h})$. The expected sum of rewards in $\mathbf{M}$, when actions a_h are chosen according to a practicable policy π, i.e., when $a_h = \pi_h(\mathbf{x}_{1:h})$, is referred to as the *value of* π *in* $\mathbf{M}$ and denoted $V^\pi_{\mathbf{M}} = \mathrm{E}_{\mathbf{M},\pi}[r_1 + r_2 + \cdots + r_H]$; the subscript in the expectation signifies the probability distribution over episode realizations when the environment follows $\mathbf{M}$, and actions are chosen according to π.

We also allow practicable policies to be (Markovian) mappings $\pi : [H] \times \mathcal{X} \to \mathcal{A}$ which choose every action according to the last observation, so that $a_h = \pi_h(x_h)$ for all h. Whether a particular practicable policy is Markovian or not will generally be clear from context.

3.2 Abstract simulator

Our learning algorithm can interact with the target environment, but it additionally has access to an *abstract simulator*, which provides an idealized and abstracted version of the target environment. Formally, an abstract simulator is an episodic MDP denoted $M^\circ = \langle \mathcal{S}, \mathcal{A}, s_{\text{init}}, H, T^\circ, R^\circ\rangle$, with the same state space, action space, start state and horizon as the latent MDP $M^\star$, but not necessarily the same transition and reward functions. Furthermore, we assume that abstract simulator is deterministic:

Assumption 1. *The abstract simulator M° is deterministic, i.e., $T^\circ_h(s_{h+1} \mid s_h, a_h) \in \{0, 1\}$.*

In order for the abstract simulator to be useful, it must approximate target environment. In this paper we assume that the target environment can be viewed as a "perturbed" version of the abstract simulator, using a notion of perturbation inspired by the concept of trembling-hand equilibria from extensive-form games [Selten, 1975]. Specifically, we say that an MDP M' is an η-perturbation of another MDP M if its dynamics can be realized by following M's dynamics while distorting agent actions according to some (unknown) "noise" distribution, referred to as ξ in the definition below, which keeps actions unchanged with probability at least $1 - \eta$:

Definition 1 (η-perturbation)**.** *We say that an MDP $M' = \langle \mathcal{S}, \mathcal{A}, s_{init}, H, T', R'\rangle$ is an η-perturbation of an MDP $M = \langle \mathcal{S}, \mathcal{A}, s_{init}, H, T, R\rangle$ if there exists a function $\xi : [H] \times \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{A})$ that satisfies $\xi_h(a \mid s, a) \geq 1 - \eta$ for all h, s, a, and such that*

$$T'_h(s' \mid s, a) = \textstyle\sum_{a'\in\mathcal{A}} T_h(s' \mid s, a')\xi_h(a' \mid s, a)$$
$$R'_h(s, a) = \textstyle\sum_{a'\in\mathcal{A}} R_h(s, a')\xi_h(a' \mid s, a)$$

for all h, s, a, s'; thus MDP M' can be viewed as following the dynamics of M in which each action a is stochastically replaced ("perturbed") according to ξ.

The set of all η-perturbations of M is denoted $\mathcal{C}(M, \eta)$.

We assume that the target environment is an η-perturbation of the abstract simulator for a value of $\eta < 0.5$. Thus, most of the time the target environment transitions after each action "as intended" (i.e., following known dynamics of the abstract simulator), but with a probability at most η it may depart from the intended action due to an inherent, but unknown stochasticity:

Assumption 2. *$M^\star$ is an η-perturbation of the abstract simulator M° for some $\eta < 0.5$.*

Abstract policy. Behavior of an idealized agent that can directly access latent state is formalized as a (Markovian) *abstract policy*. An abstract policy prescribes what action to take in each state s at a given step h, i.e., it is a mapping $\psi : [H] \times \mathcal{S} \to \mathcal{A}$; we write $\psi_h(s)$ for the action taken by ψ in step h and state s. The expected sum of rewards in an episodic MDP $M = \langle \mathcal{S}, \mathcal{A}, s_{\text{init}}, H, T, R \rangle$, when following ψ is called the *value of* ψ *in* M and denoted $V_M^\psi = \mathrm{E}_{M,\psi}[r_1 + r_2 + \cdots + r_H]$.

Robust abstract policy. Since the latent MDP in the target domain is a perturbation of the abstract simulator, we will seek to obtain policies that are robust to *any* allowed perturbation. For a given abstract simulator M° and the perturbation level η, we define a *robust abstract policy* ρ to be a policy that achieves the largest possible value under the worst-case choice of perturbation:

$$\rho = \operatorname*{argmax}_{\psi \in \Psi} \min_{M \in \mathcal{C}(M^\circ, \eta)} V_M^\psi, \tag{1}$$

where Ψ is the set of all mappings from $[H] \times \mathcal{S}$ to $\mathcal{A}$.

In a natural way, a robust abstract policy ρ can be composed with the perfect decoder $\phi^\star$ to obtain a (Markovian) practicable policy $\rho \circ \phi^\star : [H] \times \mathcal{X} \to \mathcal{A}$ mapping observations in the target environment to actions while still maximizing the worst-case reward among perturbations of M°. We aim for algorithms that find practicable policies that perform almost as well as this *robust practicable policy*.

3.3 The learning setting

We can now formally define our learning setting. A learning algorithm `Alg` in this setting receives as input a deterministic abstract simulator (episodic MDP) $M^\circ = \langle \mathcal{S}, \mathcal{A}, s_{\text{init}}, H, T^\circ, R^\circ \rangle$, meaning it receives the entire MDP represented in tabular form (or some other computationally convenient form). The algorithm is also provided with *oracle access* to a target environment (block MDP), $\mathbf{M}^\star = \langle M^\star, \mathcal{X}, q^\star \rangle$. This means that the algorithm cannot directly access $\mathbf{M}^\star$ itself, but can interact with it as an agent would, executing actions a_h, and receiving back observations x_h and rewards r_h in a sequence of episodes, as described above. Finally, `Alg` is given parameters $\epsilon > 0$, $\delta > 0$, and $\eta < 0.5$. It is assumed that $M^\star$ is an η-perturbation of M°. The assumption that we know the value of η is not essential and we discuss in Section 4.4 how it can be removed.

After interacting with $\mathbf{M}^\star$, the algorithm outputs a practicable policy π. The goal of learning is for π to have value almost as good as the robust practicable policy with high probability, that is, for $V_{\mathbf{M}^\star}^\pi \geq V_{\mathbf{M}^\star}^{\rho \circ \phi^\star} - \epsilon$ with probability at least $1 - \delta$ (where probability is over the algorithm's randomization as well as randomness in the interactions with the target environment). Furthermore, we require the number of episodes executed by the algorithm before outputting a policy π to be bounded by a polynomial in the number of actions $|\mathcal{A}|$, the horizon H, $1/\epsilon$, $1/\delta$, and $1/(1 - 2\eta)$. Note importantly that this polynomial must have no explicit dependence on the number of states $|\mathcal{S}|$ or observations $|\mathcal{X}|$ in the target environment. An algorithm that satisfies these criteria (given the stated assumptions) is said to achieve an *efficient transfer from abstract simulator*.

4 Main Algorithm

Our main contribution is an algorithm `TASID`, which achieves efficient transfer from an abstract simulator. The algorithm operates in two stages. In the first stage, it determines the robust abstract policy ρ for the provided abstract simulator via robust dynamic programming (Algorithm 1). In the second stage, it interacts with the target environment (via oracle access) in order to learn to predict the current latent state based on the history of observations and actions. The learnt decoding map is then composed with the robust abstract policy to obtain the practicable policy that is returned by the algorithm (see Algorithm 2).

4.1 Robust dynamic programming for abstract simulator

We obtain the robust abstract policy by instatiating the robust dynamic programming algorithm of Bagnell et al. [2001] and Iyengar [2005] to our specific notion of perturbation. The algorithm proceeds by filling out values of the *robust value function* $\tilde{V}$, which quantifies the largest sum of rewards achievable starting at any step h and state s, when assuming the worst-case perturbation of the input MDP M°. Specifically,

$$\tilde{V}_h(s) = \max_{\psi \in \Psi} \min_{M \in \mathcal{C}(M^\circ, \eta)} \mathrm{E}_{M,\psi}\big[r_h + r_{h+1} + \cdots + r_H \;\big|\; s_h = s\big].$$

Algorithm 1 Robust Dynamic Programming. $\texttt{RDP}(M^\circ, \eta)$

Input: An episodic MDP $M^\circ = \langle \mathcal{S}, \mathcal{A}, s_{\text{init}}, H, T^\circ, R^\circ \rangle$, perturbation level η.

1: $\tilde{V}_{H+1}(s) \leftarrow 0$ for all $s \in \mathcal{S}$
2: **for** $h = H, \ldots, 1$ **do**
3: **for** all $s \in \mathcal{S}, a \in \mathcal{A}$: $\tilde{Q}_h(s,a) \leftarrow R_h^\circ(s,a) + \sum_{s'} T_h^\circ(s' \mid s,a) \tilde{V}_{h+1}(s')$
4: **for** all $s \in \mathcal{S}$: $\tilde{V}_h(s) \leftarrow (1-\eta) \max_a \tilde{Q}_h(s,a) + \eta \min_a \tilde{Q}_h(s,a)$
5: **Return** ρ defined by $\rho_h(s) = \operatorname{argmax}_a \tilde{Q}_h(s,a)$

Algorithm 2 Transfer from Abstract Simulator using Inverse Dynamics. $\texttt{TASID}(\mathbf{M}^\star, M^\circ, \mathcal{F}, \eta, \epsilon, \delta)$

Input: Oracle access to target environment $\mathbf{M}^\star$, deterministic abstract simulator M°, optimization-oracle access to a function class $\mathcal{F} \subseteq \{\mathcal{X}^2 \to \Delta(\mathcal{A})\}$, perturbation level $\eta < 0.5$, target accuracy $\epsilon > 0$, failure probability δ.

1: Let ρ be the robust policy returned by $\texttt{RDP}(M^\circ, \eta)$
2: Let $n_D := \frac{8H^2|\mathcal{A}|^3 \ln(|\mathcal{F}|/\delta)}{\epsilon(1-2\eta)^2}$
3: Let $\pi_1(x) := \rho_1(s_{\text{init}})$ for all $x \in \mathcal{X}$ // Define practicable policy in step $h = 1$
4: **for** $h = 1, \ldots, H-1$ **do**
5: $\mathcal{D}_h \leftarrow \emptyset$ // Gather dataset $\mathcal{D}_h$ for learning "inverse dynamics" in step h
6: **for** n_D times **do**
7: Follow $\pi_{1:h-1}$ for $h-1$ steps to observe x_h
8: Take action a_h uniformly at random and observe x_{h+1}
9: $\mathcal{D}_h \leftarrow \mathcal{D}_h \cup \{(x_h, a_h, x_{h+1})\}$
10: $f_h := \arg\max_{f \in \mathcal{F}} \sum_{(x_h, a_h, x_{h+1}) \in \mathcal{D}_h} \ln f(a_h \mid x_h, x_{h+1})$ // Learn "inverse dynamics"
11: Define $\alpha_h : \mathcal{X}^2 \to \mathcal{A}$ as // Define "shadow action" decoder
$\alpha_h(x_h, x_{h+1}) = \operatorname{argmax}_{a \in \mathcal{A}} f_h(\cdot \mid x_h, x_{h+1})$
12: Define $\phi_{h+1} : \mathcal{X}^{h+1} \to \mathcal{S}$ such that // Define state decoder
$\phi_{h+1}(\mathbf{x}_{1:h+1})$ is the state s_{h+1} reached in M°, when starting in s_{init} and executing
$a'_1 = \alpha_1(x_1, x_2), \ldots, a'_h = \alpha_h(x_h, x_{h+1})$
13: Define $\pi_{h+1} : \mathcal{X}^{h+1} \to \mathcal{A}$ as // Define practicable policy
$\pi_{h+1}(\mathbf{x}_{1:h+1}) = \rho_{h+1}\big(\phi_{h+1}(\mathbf{x}_{1:h+1})\big)$
14: **return** $\pi = (\pi_1, \ldots, \pi_H)$

Similar to standard dynamic programming, the value function in robust dynamic programming can be filled out beginning with $h = H+1$, where we have $\tilde{V}_h(s) = 0$, and proceeding backward. In our case, the values $\tilde{V}_h(s)$ can be obtained from $\tilde{V}_{h+1}(s)$ using a closed-form expression (line 4), which leverages intermediate values $\tilde{Q}_h(s,a)$. Note that $\tilde{Q}$ is not quite the robust state-action value function, because it assumes that the action at step h is left unperturbed (and only considers the worst-case perturbation in the following steps). The function $\tilde{Q}$ is used to derive the robust policy (line 5).

Theorem 1. *For any abstract simulator M° and any perturbation level η, Algorithm 1 returns the robust policy $\rho = \operatorname{argmax}_{\psi \in \Psi} \min_{M \in \mathcal{C}(M^\circ, \eta)} V_M^\psi$.*

(The proof of this theorem and all other proofs in this paper are deferred to the appendix.)

4.2 Learning a decoder in the target environment

We cannot directly apply the robust abstract policy in the target environment, because the latent states are not observable. Therefore, we construct a "state decoder" in the target environment, which uses the history of observation in an episode to predict the current latent state; the state decoder is combined with the robust abstract policy to obtain the practicable policy. Formally, a (non-Markovian) state decoder is a mapping $\phi : \mathcal{X}^{\leq H} \to \mathcal{S}$. In step h of an episode, the history of previous observations $\mathbf{x}_{1:h}$ is used as an input, and the decoder predicts the latent state s_h. To construct such a state decoder, we crucially leverage the abstract simulator M°.

The abstract simulator M° is deterministic, so a specific sequence of actions $a_1, \ldots, a_h$ always leads to the same state. The latent MDP in the target environment is a perturbation of the abstract simulator. This means that when an agent takes action a_h in step h, most of the time the environment transitions according to $T^\circ(\cdot \mid s_h, a_h)$, but sometimes (with probability at most η) it transitions according to some other action. The action a_h gets replaced with some "shadow" action a'_h according to an unknown noise distribution ξ, and the latent state then transitions according to $T^\circ(\cdot \mid s_h, a'_h)$. If we knew shadow actions $a'_1, \ldots, a'_h$, we could then recover the current latent state by simulating that same sequence of actions in the abstract simulator.

To obtain shadow actions, we learn an "inverse dynamics" model, which predicts a'_h from the observations x_h and x_{h+1} (an approach also used in previous work on block MDPs). We learn a separate inverse dynamics model for each step of an episode. In step h, we sample triplets of the form (x_h, a_h, x'_h) across multiple episodes in the target environment, and then fit a model f_h for the conditional probability of a_h given x_h and x'_h. The model $f_h(a_h \mid x_h, x'_h)$ is referred to as an inverse dynamics model, because it "inverts" the dynamics represented by the transition function. We show that if we were able to obtain an exact model $f^\star_h$ of the conditional probability, then the action a with the largest probability $f^\star_h(a \mid x_h, x'_h)$ would be the correct shadow action.

As is standard in the block MDP literature, in order to fit an inverse dynamics model, we assume access to an optimization algorithm capable of fitting functions from some class $\mathcal{F} \subseteq \{\mathcal{X}^2 \to \Delta(\mathcal{A})\}$ to data; we call this algorithm an *optimization oracle for* $\mathcal{F}$. The class $\mathcal{F}$ should be sufficiently expressive to approximate the required conditional probability distribution.

We now have all the pieces required to describe our algorithm. The algorithm first constructs the robust abstract policy (line 1), and defines the practicable policy π_1 at the initial step $h = 1$, where the latent state is known to be s_{init} (line 3). The algorithm then proceeds iteratively to fill in $\pi_2, \ldots, \pi_H$. In iteration h, the algorithm first learns the inverse dynamics model f_h with the help of the optimization oracle (line 5–10). The inverse dynamics model is then used to obtain the shadow action decoder α_h (line 11), which predicts which action caused the transition from x_h to x_{h+1}. Using the shadow action decoders up to step h, we can construct a state decoder ϕ_{h+1} (line 12), which for a given history of observations $\mathbf{x}_{1:h+1}$, first predicts their corresponding shadow actions $a'_1, \ldots, a'_h$ and then uses the abstract simulator to determine the state s_{h+1} that they lead to. Finally, using the state decoder ϕ_{h+1}, we define the practicable policy at the step $h + 1$ to return the same action as the abstract robust policy would return on the decoded state (line 13).

4.3 Sample Complexity of TASID

We next provide the sample complexity analysis of TASID, showing that it indeed achieves efficient transfer from an abstract simulator. In addition to Assumptions 1 and 2, we also need to ensure that the function class $\mathcal{F}$ is expressive enough to approximate the conditional probability distribution being fitted by the inverse dynamics model. It turns out that this target probability distribution can be expressed in terms of the transition function of the block MDP $\mathbf{M}^\star$, which is the function $\mathbf{T}^\star : [H] \times \mathcal{X} \times \mathcal{A} \to \Delta(\mathcal{X})$ equal to

$$\mathbf{T}^\star_h(x' \mid x, a) \coloneqq q^\star(x' \mid s' = \phi^\star(x'))T_h\big(s' = \phi^\star(x') \;\big|\; s = \phi^\star(x), a\big).$$

Using $\mathbf{T}^\star$, we can write the exact inverse dynamics model as $f^\star_h(a \mid x, x') = \frac{\mathbf{T}^\star_h(x' \mid x, a)}{\sum_{a' \in \mathcal{A}} \mathbf{T}^\star_h(x' \mid x, a')}$.

To state our assumption, let P_h denote the distribution over triples (x_h, a_h, x_{h+1}) sampled in line 7-8, and let $\widetilde{f}_h$ be the maximizer (in $\mathcal{F}$) of the expected log likelihood under P_h:

$$\widetilde{f}_h \coloneqq \operatorname*{argmax}_{f \in \mathcal{F}} \mathrm{E}_{x,a,x' \sim P_h}[\ln f(a \mid x, x')].$$

Assumption 3 (Approximate realizability)**.** *There exists $\epsilon_{\mathcal{F}} \geq 0$ such that for every $h \in [H]$, and for all triples (x, a, x') in the support of P_h*

$$f^\star_h(a \mid x, x') \leq (1 + \epsilon_{\mathcal{F}})\widetilde{f}_h(a \mid x, x').$$

Exact realizability assumptions (with $\epsilon_{\mathcal{F}} = 0$) are standard in the block MDP literature [Du et al., 2019, Agarwal et al., 2020]. Here we only assume approximate realizability, which can be assured for any $\epsilon_{\mathcal{F}} > 0$ by choosing a sufficiently expressive class such as deep neural networks. In practice, we choose classes that express some inductive biases (such as feature invariance, factorization, etc.).

We are now ready to state our main theoretical result—sample complexity of TASID.

Theorem 2. *Let M° be a deterministic abstract simulator. Let $\mathbf{M}^\star$ be a target environment for which $M^\star$ is an η-perturbation of M° for some $\eta < 0.5$. Let $\mathcal{F}$ be a class of functions that satisfies Assumption 3 with $\epsilon_{\mathcal{F}} \leq \frac{(1-2\eta)^2}{8H^4|\mathcal{A}|^3}$. Then for any $\epsilon > 0$ and $\delta \in (0,1)$, Algorithm 2 with oracle access to $\mathbf{M}^\star$, optimization-oracle access to $\mathcal{F}$, and inputs M°, η, $\epsilon/2$, and δ, executes $n = \mathcal{O}\big(\frac{H^3|\mathcal{A}|^3 \ln(|\mathcal{F}|/\delta)}{\epsilon(1-2\eta)^2}\big)$ episodes and returns a practicable policy π that with probability at least $1-\delta$ satisfies*

$$V^{\pi}_{\mathbf{M}^\star} \geq V^{\rho\circ\phi^\star}_{\mathbf{M}^\star} - \epsilon.$$

Thus, Algorithm 2 achieves efficient transfer from an abstract simulator, meaning that its sample complexity is independent of the sizes of $|\mathcal{S}|$ and $|\mathcal{X}|$. We also achieve a fast rate, $\mathcal{O}(1/\epsilon)$, with respect to the sub-optimality ϵ. The assumption on $\epsilon_{\mathcal{F}}$ is not restrictive, because we can always choose a sufficiently expressive class. Note that previous works simply assume $\epsilon_{\mathcal{F}} = 0$ [e.g., Du et al., 2019, Agarwal et al., 2020, Misra et al., 2020]. The dependence on $\ln|\mathcal{F}|$ is standard; for neural networks it is proportional to the number of bits to store the network weights. For example, if the neural network has 1000 weights represented by doubles (i.e., with 64 bits) then $\ln|\mathcal{F}| = (\ln 2)\cdot 64000$. There may be room to improve dependence on the horizon H and action space size $|\mathcal{A}|$, but these were not our focus here.

4.4 Extensions

In the appendix, we show how TASID, under additional assumptions, can be extended to settings where the abstract simulator has a stochastic start state (but deterministic transitions).

The assumption that η is known is not necessary. If η is not known, it is possible to carry out a form of binary search as follows: Run the algorithm with $\eta = 1/4$ and verify whether it achieves the performance guarantee. If it does, continue with $\eta = 1/8$ (splitting the interval $(0, 1/4]$), otherwise continue with $\eta = 3/8$ (splitting the interval $(1/4, 1/2)$), and run the algorithm again. If the goal is to recover η up to the precision ϵ_η, then we just need to repeat this $\log_2(1/2\epsilon_\eta)$ times.

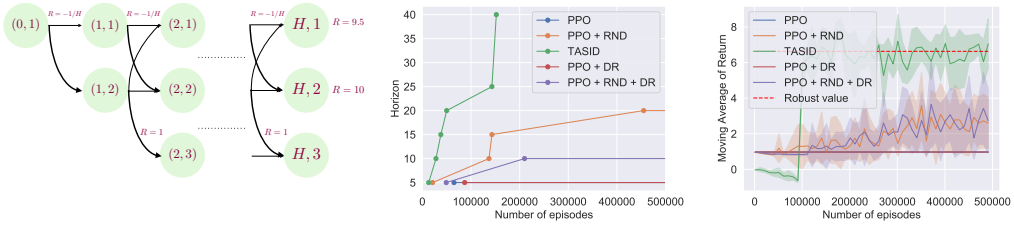
Although our focus here is on transfer from an abstract simulator, and we emphasize the relative ease of constructing abstract simulators, our approach could also be used for transfer learning between two block MDP environments that share latent space structure. In the source domain, we could use an existing block MDP approach [e.g., Misra et al., 2020] to infer latent structure, which could then be used as an abstract simulator to speed up learning in the target domain using TASID.

5 Experiments

We evaluate TASID in two simulation environments, focusing on four of its properties: sample complexity independent of state-space size; scalability to complex visual observations; robustness of the policy it learns; and scalability to large state spaces. We summarize the results here and defer the details, including hyperparameter selection, detailed environment descriptions, algorithm and baselines implementation and a full description of results under various setups to the appendix.

Can TASID solve problems that require strategic exploration? We theoretically showed that TASID can solve problems that require performing strategic exploration using a small number of episodes. We test this empirically on a challenging environment called *combination lock*.

Combination lock. We first describe the abstract simulator M° visualized in Figure 2a. It has an action space $\mathcal{A}$, horizon H, and a state space $\{(h,i) \mid h \in \{0\} \cup [H], i \in [3]\}$, with the initial state $(0,1)$. As we will see, states $\{(h,1),(h,2) \mid h \in [H]\}$ are *good* states from which optimal return is possible, while states $\{(h,3) \mid h \in [H]\}$ are *bad* states. In state $(h,1)$, one good action leads to state $(h+1,1)$, and all other $|\mathcal{A}|-1$ actions lead to state $(h+1,2)$. In state $(h,2)$, one good action leads to state $(h+1,2)$, another good action leads to state $(h+1,1)$, and all other $|\mathcal{A}|-2$ actions lead to $(h+1,3)$. All actions in state $(h,3)$ lead to state $(h+1,3)$. The identities of good actions are unknown and are different for different states. Any transition to state $(H,1)$ gives a reward of 9.5, whereas transition to state $(H,2)$ gives a reward of 10, and transition to state $(H,3)$ gives a reward of 0. To "mislead" the agent, transitions from $(h,2)$ to $(h+1,3)$ give a reward of 1 and transitions from $(h,1)$ to $(h+1,1)$ give a reward of $-1/H$. All other transitions give a reward of 0.



(a) Combination lock environment (b) Horizon solved *vs* episodes (c) Reward curve for $H = 40$

Figure 2: (a): Latent MDP in the combination lock. (b): The largest problem (expressed as horizon H) that each algorithm can solve (defined as reaching 95% of the value of the robust policy) within 500 000 episodes; we consider $H = 5, 10, 15, 20, 25, 40$ and $|\mathcal{A}| = 10$. We report median number of episodes across five trials with different seeds (see the appendix for full details). (c): Total reward per episode for the problem size $H = 40$. TASID is a batch algorithm trained on the first 100 000 episodes, thus its performance before the end of training is very low and after the end of training stays constant. Shaded bands correspond to the 95% Student's t confidence intervals (here and elsewhere in paper).

The latent MDP $M^\star$ of the target environment is constructed by taking a random η-perturbation of M° for $\eta = 0.1$. Exploration in both $M^\star$ and M° can be difficult due to misleading rewards and challenging dynamics, where most actions lead to bad states. The optimal policy in M° finishes in the state $(H, 2)$ and achieves the value of 10. However, the robust policy will attempt to visit the state $(H, 1)$ which lies on a more stable trajectory. Meanwhile, the optimal policy in M° will fail in $M^\star$, because perturbations are likely to move the agent into a bad state.

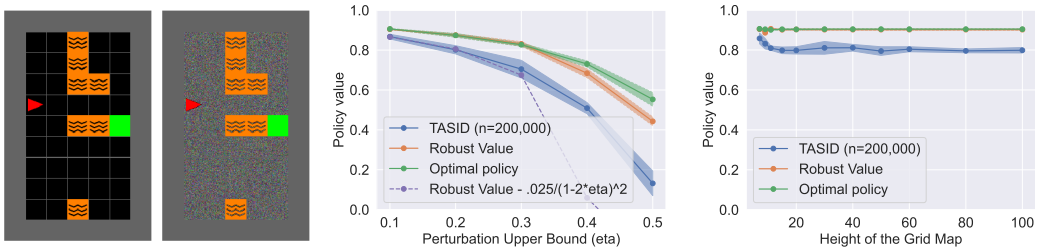
Observations in the target environment are real-valued vectors of dimension $2^{\lceil \log_2(H+4) \rceil}$. For a latent state (h, i), the observation is generated by first creating an $(H + 4)$-dimensional vector by concatenating one-hot encodings of h and i, element-wise adding Gaussian noise, applying a fixed coordinate permutation, and finally multiplying the vector with a Hadamard matrix.

We compare TASID with four baselines. The first is PPO [Schulman et al., 2017], a policy-gradient-based algorithm. The second is PPO-RND, which augments PPO with an exploration bonus based on prediction error [Burda et al., 2019b]. These two baselines do not use abstract simulator and run on the target environment from scratch. The other two baselines, PPO+DR and PPO+RND+DR, enhance PPO and PPO+RND with domain randomization (DR), where the policy is pre-trained on a set of randomized block MDPs [Tobin et al., 2017]. PPO+DR and PPO+RND+DR are designed for transfer RL, but unlike our work, they rely on observation-based similarity. During the pre-training phase of domain randomization, we generate block MDPs by following a similar process as the one used to generate the target environment. However, when generating the emission function, we only generate a random permutation matrix and the other components are kept the same as in the target environment. This randomization set includes the true emission function and transition dynamics.

For baseline algorithms, we run grid search over hyperparameters listed in Table 3 in Appendix D, separately for each environment specification (each value of H), and report the best results of PPO(+RND)(+DR). For TASID, we consider only one hyperparameter, the number of training episodes per time step n_D, and search over three possible values: 1000, 2500, 10000.

Figure 2b plots the size of the problem (expressed as horizon H) that each algorithm can solve (defined as reaching at least 95% of the value of the robust policy) within a given number of episodes; we considered $H = 5, 10, 15, 20, 25, 40$. The plot shows that PPO and PPO+DR fail to solve the problem beyond the smallest size ($H = 5$), which is not surprising as they are not designed to perform strategic exploration. PPO+RND can solve the problem until $H = 20$, but fails for $H = 25$, showing that RND bonus helps to an extent but cannot solve harder problems. Interestingly, PPO+RND+DR underperforms compared to PPO+RND even though it has access to pre-training. We believe this is because RND reward bonus does not work well with domain randomization which can mislead the RND bonus by randomizing the observation. Finally, TASID can solve the problem for all values of H and is more sample efficient than the baselines.

Can TASID scale to complex observation spaces? We evaluate TASID in the visual MiniGrid environment [Chevalier-Boisvert et al., 2018] with noisy observations. We test on the grid world



(a) Minigrid (b) Noise (c) Performance *vs* perturbation (d) Performance *vs* state space size

Figure 3: (a,b): Map of a 7×11 MiniGrid environment with noisy observations and five actions: forward, turn left, turn right, turn left + forward, turn right + forward. (c): The performance of `TASID` with different amount of perturbation η. (d): The performance of `TASID` with different heights of the map.

map shown in Figure 3a. The green square represents the goal and orange squares are lava that should be avoided. The agent's position and direction is shown by the red triangle. The agent gets a reward of 1 on reaching the goal, -1 for reaching the lava, and -0.01 for every other step. The state encodes the position and direction of the agent and the current time step. There are 5 actions: move forward, turn left, turn right, and combinations of turning and moving. The abstract simulator M° models the deterministic transitions on the grid. The target environment is an η-perturbation of M° for $\eta = 0.1$. The agent cannot observe the whole map as in Figure 3a, but only an image of the 7×7 grids (56×56 pixels) in the direction it is facing, with i.i.d. random noise added to all black pixels as shown in Figure 3b. This type of random noise has been studied in the literature and has been shown to pose challenge for RL algorithms [Burda et al., 2019a].

In this grid world, the agent can either use the top route to reach the goal, or a bottom route. The top route is shorter but the agent is more likely to visit the lava due to perturbations. Therefore, the robust policy prefers the bottom route which is longer but more robust.

How robust is `TASID` to different η? Our theoretical analysis suggests that the suboptimality scales as $\mathcal{O}(1/(1-2\eta)^2)$. We evaluate this error empirically in the grid world environment for various values of η in $[0.1, 0.5]$ and show the results in Figure 3c. The performance matches the theoretical prediction of $V^{\rho \circ \phi^\star} - \mathcal{O}(1/(1-2\eta)^2)$ until $\eta = 0.3$, and becomes even better for larger η. Interestingly, even though the theoretical guarantee is vacuous at $\eta = 0.5$, the algorithm still learns a non-trivial policy with the value greater than that of a random policy. Note that we tune the value of n_D rather than use the value suggested by our theory.

How does `TASID`'s performance scale with $|\mathcal{S}|$? We showed theoretically that `TASID`'s sample complexity to learn a near-optimal robust policy is independent of the size of the state space. We empirically test this by varying the height of the grid world environment while keeping the width and the horizon H, and overall layout the same. Results are presented in Figure 3d. As height changes from 10 to 100, the size of the state space increases ten times; however, we do not see a significant drop in the performance of `TASID` when using a fixed number of episodes in the target environment.

6 Conclusion and Future Work

We presented a new algorithm `TASID` for transfer RL in block MDPs that quickly learns a robust policy in the target environment by leveraging an abstract simulator. We have also proved a sample complexity bound, which does not scale with the size of the state space or the observation space. Finally, we demonstrated theoretical properties by empirical evaluation in two domains.

Our work raises many important questions. Our theoretical analysis depends on the assumption that the abstract simulator is deterministic and the the target environment is its perturbation. Is it possible to relax these requirements and still obtain sample bounds that do not scale with the state space size? One direction might be to leverage "full simulators" (including observations), and incorporate observation similarity between the simulator and the real environment, along with similarity in the latent dynamics. Another important question is how to extend the presented approach to continuous control problems in robotics, which are natural domains for sim-to-real transfer.

References

Alekh Agarwal, Sham Kakade, Akshay Krishnamurthy, and Wen Sun. FLAMBE: Structural complexity and representation learning of low rank MDPs. In *NeurIPS*, pages 20095–20107, 2020.

J Andrew Bagnell, Andrew Y Ng, and Jeff G Schneider. Solving uncertain Markov decision processes. Technical report, Carnegie Mellon University, 2001.

Tim Brys, Anna Harutyunyan, Matthew E. Taylor, and Ann Nowé. Policy transfer using reward shaping. In *AAMAS*, pages 181–188, 2015.

Yuri Burda, Harri Edwards, Deepak Pathak, Amos Storkey, Trevor Darrell, and Alexei A Efros. Large-scale study of curiosity-driven learning. In *ICLR*, 2019a.

Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. *ICLR*, 2019b.

Maxime Chevalier-Boisvert, Lucas Willems, and Suman Pal. Minimalistic gridworld environment for OpenAI gym. `https://github.com/maximecb/gym-minigrid`, 2018.

Simon Du, Akshay Krishnamurthy, Nan Jiang, Alekh Agarwal, Miroslav Dudik, and John Langford. Provably efficient RL with rich observations via latent state decoding. In *ICML*, pages 1665–1674. PMLR, 2019.

Fernando Fernández, Javier García, and Manuela Veloso. Probabilistic policy reuse for inter-task transfer learning. *Robot. Auton. Syst.*, 58(7):866–871, 2010.

Garud N Iyengar. Robust dynamic programming. *Mathematics of Operations Research*, 30(2): 257–280, 2005.

Nan Jiang, Akshay Krishnamurthy, Alekh Agarwal, John Langford, and Robert E Schapire. Contextual decision processes with low Bellman rank are PAC-learnable. In *ICML*, pages 1704–1713. PMLR, 2017.

Sham Machandranath Kakade. *On the sample complexity of reinforcement learning*. PhD thesis, University College London, 2003.

Akshay Krishnamurthy, Alekh Agarwal, and John Langford. PAC reinforcement learning with rich observations. *NeurIPS*, 2016.

Alessandro Lazaric. Transfer in reinforcement learning: A framework and a survey. In Marco Wiering and Martijn van Otterlo, editors, *Reinforcement Learning: State-of-the-Art*, pages 143–173. Springer, 2012.

Yaxin Liu and Peter Stone. Value-function-based transfer for reinforcement learning using structure mapping. In *AAAI*, pages 415–420, 2006.

Timothy A Mann and Yoonsuck Choe. Directed exploration in reinforcement learning with transferred knowledge. In *EWRL*, pages 59–76. PMLR, 2013.

Dipendra Misra, Mikael Henaff, Akshay Krishnamurthy, and John Langford. Kinematic state abstraction and provably efficient rich-observation reinforcement learning. In *ICML*, pages 6961–6971. PMLR, 2020.

Aravind Rajeswaran, Sarvjeet Ghotra, Balaraman Ravindran, and Sergey Levine. EPOpt: Learning robust neural network policies using model ensembles. *ICLR*, 2017.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Reinhard Selten. A reexamination of the perfectness concept for equilibrium points in extensive games. *International Journal of Game Theory*, 4(1):25–55, 1975.

Vishal Soni and Satinder Singh. Using homomorphisms to transfer options across continuous reinforcement learning domains. In *AAAI*, pages 494–499, 2006.

Yanchao Sun, Ruijie Zheng, Xiyao Wang, Andrew Cohen, and Furong Huang. Transfer RL across observation feature spaces via model-based regularization. *ICLR*, 2022.

Matthew E. Taylor and Peter Stone. Behavior transfer for value-function-based reinforcement learning. In *AAMAS*, pages 53–59, 2005.

Matthew E. Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *JMLR*, 10:1633–1685, 2009.

Matthew E Taylor, Peter Stone, and Yaxin Liu. Transfer learning via inter-task mappings for temporal difference learning. *JMLR*, 8, 2007a.

Matthew E. Taylor, Shimon Whiteson, and Peter Stone. Transfer via inter-task mappings in policy search reinforcement learning. In *AAMAS*, 2007b.

Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *IROS*, pages 23–30. IEEE, 2017.

Geoffrey Van Driessel and Vincent Francois-Lavet. Component transfer learning for deep RL based on abstract representations. *arXiv preprint arXiv:2111.11525*, 2021.

Amy Zhang, Clare Lyle, Shagun Sodhani, Angelos Filos, Marta Kwiatkowska, Joelle Pineau, Yarin Gal, and Doina Precup. Invariant causal prediction for block MDPs. In *ICML*, pages 11214–11224. PMLR, 2020.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See Section 6.
   (c) Did you discuss any potential negative societal impacts of your work? [N/A] Not applicable. It is mainly a theoretical work.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes] In appendices.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] In appendices.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] In appendices.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] We use confidence interval from Student's t test.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] In appendices.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] In appendix.
   (b) Did you mention the license of the assets? [Yes] In appendix.
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] In supplementary.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A] We did not use this type of data.
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

Notation	Definition
$[N]$	For any natural number $N \in \mathbb{N}$, denotes the set $\{1, 2, \cdots, N\}$.
$\lvert\mathcal{U}\rvert$	Cardinality of a set $\mathcal{U}$. E.g., $\lvert[N]\rvert = N$.
$\Delta(\mathcal{U})$	For any countable set $\mathcal{U}$, denotes the set of all probability distribution over $\mathcal{U}$
$\texttt{unf}(\mathcal{U})$	Denotes the uniform distribution over a set $\mathcal{U}$.
$\mathcal{S}$	A finite but potentially very large state space.
$\mathcal{A}$	Agent's finite action space.
H	Horizon denoting the number of actions the agent can take in an episode.
T	For a given $h \in [H], a \in \mathcal{A}$ and states $s, s' \in \mathcal{S}$, $T_h(s' \mid s, a)$ denotes the probability of transitioning to s' when taking action a in state s at time step h.
R	For a given $h \in [H], a \in \mathcal{A}$ and state $s \in \mathcal{S}$, $R_h(s, a)$ denotes the reward for taking action a in state s at time step h.
s_{init}	A deterministic initial state
M	Markov Decision Process (MDP) defined by a tuple $(S, \mathcal{A}, s_{\text{init}}, H, T, R)$.
$\mathcal{X}$	Agent's observation space. Can be potentially infinitely large.
$q : \mathcal{S} \to \Delta(\mathcal{X})$	Emission function where for a given $x \in \mathcal{X}$ and $s \in \mathcal{S}$, $q(x \mid s)$ denotes the probability of observing x when the agent is in latent state s
$\mathbf{M}$	A block MDP defined by the tuple $(M, \mathcal{X}, q)$ where M is referred to as the latent MDP since the agent receives observation instead of state which remains latent.
M°	Abstract simulator which is a Markov Decision Process (MDP) given by $(\mathcal{S}, \mathcal{A}, s_{\text{init}}, H, T^\circ, R^\circ)$ and satisfies Assumption 1.
$\mathbf{M}^\star$	Target block MDP defined by $(M^\star, \mathcal{X}, q)$ where the target environment's latent MDP is given by $(\mathcal{S}, \mathcal{A}, s_{\text{init}}, H, T^\star, R^\star)$. The latent MDP $M^\star$ satisfies Assumption 2.
$\phi^\star : \mathcal{X} \to \mathcal{S}$	Perfect decoder of the latent state for $\mathbf{M}^\star$. Agent does not have access to this.
$\pi : \mathcal{X}^{\leq H} \to \mathcal{A}$	Non-Markovian practicable policy. Given an observation sequence $\mathbf{x}_{1:h}$, the policy takes action $\pi_h(\mathbf{x}_{1:h})$.
$V^\pi_{\mathbf{M}}$	Value of a practicable policy π in a block MDP $\mathbf{M}$.
$\psi : [H] \times \mathcal{S} \to \mathcal{A}$	A non-stationary Markovian abstract policy. Given state s at time step h, the policy takes action $\psi_h(s)$.
Ψ	Set of all latent policies, i.e., all mappings of the type $[H] \times \mathcal{S} \to \mathcal{A}$.
V^ψ_M	Value of abstract policy ψ in MDP M.
η	Perturbation parameter controlling the amount of stochasticity in the target environment. $\eta = 0$ implies deterministic dynamics.
$C(M, \eta)$	Set of all MDPs which are η-perturbations of M.
ρ	Robust policy defined Equation 1,
ϵ	Allowed error in learning the robust practicable policy.
δ	Allowed failure probability in learning the robust practicable policy.
$\tilde{V}_h(s)$	Robust value function. See Section 4.1.
$\mathcal{F}$	A function class containing functions of type $\mathcal{X}^2 \to \Delta(\mathcal{X})$, used by `TASID` to learn inverse dynamics model.
$\alpha_h : \mathcal{X}^2 \to \mathcal{A}$	Learned "shadow action" decoder in `TASID` for time step h.
$\phi_h : \mathcal{X}^h \to \mathcal{S}$	Learned state decoder in `TASID` for time step h.
$\mathcal{D}_h$	Dataset of transitions collected by `TASID` at time step h.

Table 1: A list of notations.

In the appendices, we include proofs, extended theoretical results as well as experimental details for all claims in the main paper, under the following structure.

- We provide a list of notations in Table 1.
- Appendix A includes the proof for the analysis of Algorithm 1.
- Appendix B includes the proof for the analysis of Algorithm 2.
- Appendix C includes the algorithm and analysis for the case of stochastic initial states, based on the main algorithm.
- Appendix D includes more details of the experiments.

A Proofs for Algorithm 1

We introduce the definition of the (single-step) perturbed transition and reward set, following the definition of $\mathcal{C}(M^\circ, \eta)$.

Definition 2 (η-perturbation)**.** *We say that a transition and reward function pair (T', R') at step h is an η-*perturbation *of the pair (T_h, R_h) if there exists a function $\xi : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{A})$ that for all s, s' and a satisfies $\xi(a \mid s, a) \geq 1 - \eta$ and*

$$T'(s' \mid s, a) = \textstyle\sum_{a' \in \mathcal{A}} T_h(s' \mid s, a')\xi(a' \mid s, a)$$
$$R'(s, a) = \textstyle\sum_{a' \in \mathcal{A}} R_h(s, a')\xi(a' \mid s, a).$$

With slight abusing of the notation, the set of all η-perturbations of T_h, R_h is denoted $\mathcal{C}((R_h, T_h), \eta)$.

It is straightforward to see that $(R', T') \in \mathcal{C}((R_h, T_h), \eta)$ if and only if there exists an MDP $M' \in \mathcal{C}(M, \eta)$ such that (R', T') is the h-th step reward and transition functions in M' and (R_h, T_h) is the h-th step reward and transition functions in M.

Now we prove a lemma about the function $\tilde{Q}_h$ used in Algorithm 1.

Lemma 1. *For any $h \in [H]$, and $s \in \mathcal{S}_h$, the learned function $\tilde{Q}_h$ in Algorithm 1 satisfies:*

$$(1-\eta) \max_{a \in \mathcal{A}} \tilde{Q}_h(s, a) + \eta \min_{a \in \mathcal{A}} \tilde{Q}_h(s, a) = \max_{a \in \mathcal{A}} \min_{(R,T) \in \mathcal{C}((R_h^\circ, T_h^\circ), \eta)} \mathrm{E}_{s' \sim T(\cdot|s,a)}[R(s, a) + \tilde{V}_{h+1}(s')]$$

Proof. By the definition of $\mathcal{C}((R_h^\circ, T_h^\circ), \eta)$, we know that

$$\min_{(R,T) \in \mathcal{C}((R_h^\circ, T_h^\circ), \eta)} \mathrm{E}_{s' \sim T(\cdot|s,a)}[R(s, a) + \tilde{V}_{h+1}(s')] \tag{2}$$
$$= \min_{(R,T) \in \mathcal{C}((R_h^\circ, T_h^\circ), \eta)} R(s, a) + \sum_{s' \in \mathcal{S}} T(s'|s, a)\tilde{V}_{h+1}(s') \tag{3}$$
$$= \min_{\xi(\cdot|s,a)} \sum_{a' \in \mathcal{A}} \left(\xi(a'|s, a)R_h^\circ(s, a') + \sum_{s' \in \mathcal{S}} \xi(a'|s, a)T_h^\circ(s'|s, a)\tilde{V}_{h+1}(s') \right) \tag{4}$$
$$= \min_{\xi(\cdot|s,a)} \sum_{a' \in \mathcal{A}} \xi(a'|s, a) \left(R_h^\circ(s, a') + \sum_{s' \in \mathcal{S}} T_h^\circ(s'|s, a)\tilde{V}_{h+1}(s') \right) \tag{5}$$
$$= \min_{\xi(\cdot|s,a)} \sum_{a'} \xi(a'|s, a)\tilde{Q}_h(s, a') \tag{6}$$
$$=(1-\eta)\tilde{Q}_h(s, a) + \eta \min_{a' \in \mathcal{A}} \tilde{Q}_h(s, a'), \tag{7}$$

where $\min$ over $\xi(\cdot|s, a)$ is taken so that it still satisfies the requirement of $\xi(a \mid s, a) \geq 1 - \eta$. This is then used in the last equality, where the minimum is achieved by placing the maximum allowed mass, which is η, on the action with smallest value of $\tilde{Q}_h(s, a')$ and remaining on $\tilde{Q}_h(s, a)$. Thus we proved that

$$(1-\eta)\tilde{Q}_h(s, a) + \eta \min_{a'} \tilde{Q}_h(s, a') = \min_{(R,T) \in \mathcal{C}((R_h^\circ, T_h^\circ), \eta)} \mathrm{E}_{s' \sim T(\cdot|s,a)}[R(s, a) + \tilde{V}_{h+1}(s')]$$

The proof is completed by taking maximum over actions on both side. □

Completing the Proof of Theorem 1

Now we prove the Theorem 1 as the main result for Algorithm 1.

Proof. By Theorem 2.2 in Iyengar [2005], since Q_h and V_h satisfy Lemma 1, we have that $\tilde{V}_1(s) = \sup_{\psi \in \Psi} \min_{M \in \mathcal{C}(M^\circ, \eta)} V_{M,1}^{\psi}(s)$. The perturbation we defined consists of independent perturbation at different time step h, therefore it satisfy the "rectangularity" condition defined in Iyengar [2005].[2]

[2] Rectangularity condition requires the choice of perturbation in one time step cannot limit the choice of perturbation in other time steps. We refer to Iyengar [2005] for the formal definition.

From Iyengar [2005], the robust policy is given by

$$\rho_h(s) = \operatorname*{argmax}_a \min_{(R,T)\in\mathcal{C}((R_h^\circ,T_h^\circ),\eta)} \mathrm{E}_{s'\sim T(\cdot|s,a)}[R(s,a)+\tilde{V}_{h+1}(s')] \tag{8}$$

$$= \operatorname*{argmax}_a \left((1-\eta)\tilde{Q}_h(s,a) + \eta \min_{a'\in\mathcal{A}} \tilde{Q}_h(s,a')\right) \tag{9}$$

$$= \operatorname*{argmax}_{a\in\mathcal{A}} \tilde{Q}_h(s,a) \tag{10}$$

This is how Algorithm 1 computes the policy it returns. Therefore, Algorithm 1 returns the robust policy ρ. □

A.1 An alternative Proof for Algorithm 1 from the First Principles

The previous proof relies on a more general results from Iyengar [2005] in a more complicated form. However the proof can be simplified since our perturbation set takes a particular linear structure. So for completeness and readability, we also include another complete proof derived from the first principle. We acknowledge that the proof is heavily inspired by previous work in robust dynamics programming [Bagnell et al., 2001, Iyengar, 2005].

Let Ψ be the set of all mappings from $[H]\times\mathcal{S}$ to $\Delta(\mathcal{A})$. The following proposition follows directly from application of max-min inequality.

Proposition 1. $\forall h\in[H], s\in\mathcal{S}$

$$\max_{\psi\in\Psi}\min_{M\in\mathcal{C}(M^\circ,\eta)} V_{M,h}^{\psi}(s) \le \min_{M\in\mathcal{C}(M^\circ,\eta)}\max_{\psi\in\Psi} V_{M,h}^{\pi}(s) \coloneqq \min_{M\in\mathcal{C}(M^\circ,\eta)} V_{M,h}^{\star}(s)$$

Given this, we are going to explain the high level structure of the proof. For $\tilde{V}_h$, we are going to show that

$$\tilde{V}_h(s) \ge \min_{M\in\mathcal{C}(M^\circ,\eta)} V_{M,h}^{\star}(s) \tag{11}$$

$$\tilde{V}_h(s) \le \min_{M\in\mathcal{C}(M^\circ,\eta)} V_{M,h}^{\tilde{\psi}}(s), \tag{12}$$

for some $\tilde{\psi}$.

Then from Proposition 1, we will have that

$$\tilde{V}_h(s) \le \min_{M\in\mathcal{C}(M^\circ,\eta)} V_{M,h}^{\tilde{\psi}}(s) \le \max_{\psi\in\Psi}\min_{M\in\mathcal{C}(M^\circ,\eta)} V_{M,h}^{\psi}(s) \le \min_{M\in\mathcal{C}(M^\circ,\eta)} V_{M,h}^{\star}(s) \le \tilde{V}_h(s) \tag{13}$$

Then we will have that all the inequalities must be equality and the $\tilde{\psi}$ must be the robust policy ρ.

Now we will show that Equation 11 and Equation 12 holds for the $\tilde{\psi}$ which is the output from Algorithm 1, in the following two lemmas.

Lemma 2. *For any $s\in\mathcal{S}$ and $h\in[H]$, $\tilde{V}_h(s)\ge\min_{M\in\mathcal{C}(M^\circ,\eta)} V_{M,h}^{\star}(s)$*

Proof. For any $h\in[H]$, we will construct a $M\in\mathcal{C}(M^\circ,\eta)$ such that $\forall s, \tilde{V}_h(s) = V_{M,h}^{\star}(s)$. Then for any $s\in\mathcal{S}$, $\tilde{V}_h(s) = V_{M,h}^{\star}(s) \ge \min_{M\in\mathcal{C}(M^\circ,\eta)} V_{M,h}^{\star}(s)$ and we prove this lemma. Now we prove the following statement by induction on h. For any $h\in[H+1]$, $\exists M\in\mathcal{C}(M^\circ,\eta)$ such that $\forall s\in\mathcal{S}, \tilde{V}_h(s) = V_{M,h}^{\star}(s)$.

For the basement of induction, $\tilde{V}_{H+1}(s) = V_{M,h+1}^{\star}(s) = 0$ for any M. Thus the induction statement holds for $h = H+1$.

Second, let us assume that the induction statement holds for $h+1$, where $h\in[H]$: $\forall s, \tilde{V}_{h+1}(s) = V_{M_{h+1},h+1}^{\star}(s)$ for some $M_{h+1}\in\mathcal{C}(M^\circ,\eta)$. Notice that this statement is for the value function at $h+1$ step and holds for M_{h+1}, thus for any $M\in\mathcal{C}(M^\circ,\eta)$ that shares the reward and transition functions on and after step $h+1$, the statement also holds. This is because in our definition of episodic MDP the perturbation class the reward and transition functions at different steps are independent.

Now we are going to construct M_h such that $\tilde{V}_h(s) = V^\star_{M_h,h}(s)$. More specifically, we will only construct the reward and transition functions at h-th step: R_h and T_h, and concatenate it with other component in M_{h+1}.

For any s, let $\underline{a}$ be $\operatorname{argmin}_{a\in\mathcal{A}} \tilde{Q}_h(s,a)$. Given the s and $\underline{a}$, construct $T_h(s'|s,a)$ and $R_h(s,a)$ for any s, a, s' as the following.

$$T_h(s'|s,a) = (1-\eta)T^\circ_h(s'|s,a) + \eta T^\circ_h(s'|s,\underline{a}) \tag{14}$$
$$R_h(s,a) = (1-\eta)R^\circ_h(s,a) + \eta R^\circ_h(s,\underline{a}) \tag{15}$$

Then we finish the proof of statement for h by

$$\begin{aligned}
\tilde{V}_h(s) =&(1-\eta)\max_a \tilde{Q}_h(s,a) + \eta \min_a \tilde{Q}_h(s,a) && (16)\\
=&(1-\eta)\max_a \left(R^\circ_h(s,a) + \sum_{s'} T^\circ_h(s'|s,a)\tilde{V}_{h+1}(s')\right) && (17)\\
&+\eta\left(R^\circ_h(s,\underline{a}) + \sum_{s'} T^\circ_h(s'|s,\underline{a})\tilde{V}_{h+1}(s')\right)\\
=&\max_a \left(R_h(s,a) + (1-\eta)\sum_{s'} T^\circ_h(s'|s,a)\tilde{V}_{h+1}(s') + \eta\sum_{s'} T^\circ_h(s'|s,\underline{a})\tilde{V}_{h+1}(s')\right) && (18)\\
=&\max_a \left(R_h(s,a) + \sum_{s'} \left((1-\eta)T^\circ_h(s'|s,a) + \eta T^\circ_h(s'|s,\underline{a})\right)\tilde{V}_{h+1}(s')\right) && (19)\\
=&\max_a \left(R_h(s,a) + \sum_{s'} T_h(s'|s,a)\tilde{V}_{h+1}(s')\right) && (20)\\
=&\max_a \left(R_h(s,a) + \sum_{s'} T_h(s'|s,a)V^\star_{M_{h+1},h+1}(s)\right) && (21)\\
=&V^\star_{M_h,h}(s) && (22)
\end{aligned}$$

By induction, we proved that there exist a $M \in \mathcal{C}(M^\circ,\eta)$ such that $\tilde{V}_h(s) = V^\star_{M,h}(s)$ for any $s \in \mathcal{S}, h \in [H+1]$. Thus we finish the proof of $\tilde{V}_h(s) \geq \min_{M\in\mathcal{C}(M^\circ,\eta)} V^\star_{M,h}(s)$. □

Lemma 3. *For the policy $\tilde{\psi}$ computed from Algorithm 1, $\forall h \in [H], s \in \mathcal{S}, \min_{M\in\mathcal{C}(M^\circ,\eta)} V^{\tilde{\psi}}_{M,h}(s) \geq \tilde{V}_h(s)$*

Proof. We prove it by induction on h from $H+1$ to 1. In the base case ($h = H+1$) we have, $\tilde{V}_{H+1}(s) = 0 = V^{\tilde{\psi}}_{M,H+1}(s)$ for all $s \in \mathcal{S}$ and any M. This implies: $\forall s, \tilde{V}_{H+1}(s) \leq \min_{M\in\mathcal{C}(M^\circ,\eta)} V^{\tilde{\psi}}_{M,H+1}(s)$.

We now consider the general case. We assume that $\forall s \in \mathcal{S}, \tilde{V}_{h+1}(s) \leq \min_{M\in\mathcal{C}(M^\circ,\eta)} V^{\tilde{\psi}}_{M,h+1}(s)$. Fix any $s \in \mathcal{S}$ and any $M \in \mathcal{C}(M^\circ,\eta)$, we are going to show that $\tilde{V}_h(s) \leq V^{\tilde{\psi}}_{M,h}(s)$. Let R_h and T_h be its reward and transition function at h-th step, and ξ_h be the perturbation variables. We have $1-\xi_h(a \mid s,a) \leq \eta$ by the definition of $M \in \mathcal{C}(M^\circ,\eta)$. So $1-\xi_h(\tilde{\psi}(s) \mid s, \tilde{\psi}(s)) \leq \eta$.

$$\begin{aligned}
\tilde{V}_h(s) =&(1-\eta)\max_a \tilde{Q}_h(s,a) + \eta \min_a \tilde{Q}_h(s,a) && (23)\\
=&(1-\eta)\left(R^\circ_h(s,\tilde{\psi}(s)) + \sum_{s'} T^\circ_h(s'|s,\tilde{\psi}(s))\tilde{V}_{h+1}(s')\right) && (24)\\
&+\eta\left(R^\circ_h(s,\underline{a}) + \sum_{s'} T^\circ_h(s'|s,\underline{a})\tilde{V}_{h+1}(s')\right) && (25)
\end{aligned}$$

$$\leq \xi_h(\tilde{\psi}(s) \mid s, \tilde{\psi}(s)) \left(R_h^\circ(s, \tilde{\psi}(s)) + \sum_{s'} T_h^\circ(s'|s, \tilde{\psi}(s)) \tilde{V}_{h+1}(s') \right)$$

$$+ \left(\sum_{a \neq \tilde{\psi}(s)} \xi_h(a \mid s, \tilde{\psi}(s)) \right) \left(R_h^\circ(s, \underline{a}) + \sum_{s'} T_h^\circ(s'|s, \underline{a}) \tilde{V}_{h+1}(s') \right) \tag{26}$$

$$\leq \xi_h(\tilde{\psi}(s) \mid s, \tilde{\psi}(s)) \left(R_h^\circ(s, \tilde{\psi}(s)) + \sum_{s'} T_h^\circ(s'|s, \tilde{\psi}(s)) \tilde{V}_{h+1}(s') \right)$$

$$+ \sum_{a \neq \tilde{\psi}(s)} \xi_h(a \mid s, \tilde{\psi}(s)) \left(R_h^\circ(s, a) + \sum_{s'} T_h^\circ(s'|s, a) \tilde{V}_{h+1}(s') \right) \tag{27}$$

$$= \sum_a \xi_h(a \mid s, \tilde{\psi}(s)) \left(R_h^\circ(s, a) + \sum_{s'} T_h^\circ(s'|s, a) \tilde{V}_{h+1}(s') \right) \tag{28}$$

$$= R_h(s, \tilde{\psi}(s)) + \sum_{s'} T_h(s'|s, \tilde{\psi}(s)) \tilde{V}_{h+1}(s') \tag{29}$$

$$\leq R_h(s, \tilde{\psi}(s)) + \sum_{s'} T_h(s'|s, \tilde{\psi}(s)) \min_{M' \in \mathcal{C}(M^\circ, \eta)} V_{M', h+1}^{\tilde{\psi}}(s') \tag{30}$$

$$\leq R_h(s, \tilde{\psi}(s)) + \sum_{s'} T_h(s'|s, \tilde{\psi}(s)) V_{M, h+1}^{\tilde{\psi}}(s') = V_{M,h}^{\tilde{\psi}}(s) \tag{31}$$

Since this is for any $M \in \mathcal{C}(M^\circ, \eta)$, we proved that $\tilde{V}_h(s) \leq \min_{M \in \mathcal{C}(M^\circ, \eta)} V_{M,h}^{\pi}(s)$. □

Now we prove the Theorem 1 as the main result for Algorithm 1.

Proof. We have proved Equation 11 and Equation 12 holds for the $\tilde{\psi}$ which is the output from Algorithm 1, in previous two lemmas. Now combine this with the inequality of min max values, we have that

$$\tilde{V}_h(s) \leq \min_{M \in \mathcal{C}(M^\circ, \eta)} V_{M,h}^{\tilde{\psi}}(s) \leq \max_{\psi \in \Psi} \min_{M \in \mathcal{C}(M^\circ, \eta)} V_{M,h}^{\psi}(s) \leq \min_{M \in \mathcal{C}(M^\circ, \eta)} V_{M,h}^{\star}(s) \leq \tilde{V}_h(s) \tag{32}$$

So we will have that all the inequalities must be equality and the $\tilde{\psi}$ output by Algorithm 1 must be the robust policy ρ defined in Equation 1. □

B Proofs for Algorithm 2

In this section, we are going to prove a less restrictive and more general version of Theorem 2 stated below. We allow the $\epsilon_{\mathcal{F}}$ to take any values and the initial state in the true environment has a small probability ϵ_0 when it is not equals to the s_{init}. We show the how the error bounds scales with these approximation errors.

Theorem 3. *Let M° be a deterministic abstract simulator. Let $\mathbf{M}^\star$ be a target environment for which $M^\star$ is an η-perturbation of M° for some $\eta < 0.5$. Let ϵ_0 be the probability that $s_0 \neq s_{init}$ in M. Let $\mathcal{F}$ be a class of functions realizable with respect to $\mathbf{M}^\star$. Then for any $\epsilon > 0$ and $\delta \in (0, 1)$, Algorithm 2 with oracle access to $\mathbf{M}^\star$, optimization-oracle access to $\mathcal{F}$, and inputs M°, η, ϵ, and δ, executes $n = \mathcal{O}\big(\frac{H^4 |\mathcal{A}|^3 \ln(|\mathcal{F}|/\delta)}{\epsilon(1-2\eta)^2}\big)$ [3] episodes and returns a practicable policy π that with probability at least $1 - \delta$ satisfies*

$$V_{\mathbf{M}^\star}^{\pi} \geq V_{\mathbf{M}^\star}^{\rho \circ \phi^\star} - \epsilon - \frac{4H^4 |\mathcal{A}|^3 \epsilon_{\mathcal{F}}}{(1-2\eta)^2} - H\epsilon_0.$$

The proof to Theorem 3 has three steps. First we show the accuracy of learned ERM classifier of the classification problem in line 10. Second we show the accuracy of the learned action decoder

[3] Notice that here the dependency on horizon is H^4 instead of H^3 in the main paper. We made a mistake in the statement of this theorem in the main paper.

$\alpha_{h+1}(x_h, x_{h+1})$ under the roll-in distribution of x_h. Last the error bound in the theorem can be proved by bounding the union probability of failing to predict the action in each steps. Now we give some lemmas and proofs in these three steps in the following three subsections, followed by the proof of the final sample complexity results.

To prove Theorem 2 in the main paper, we set the ϵ in Algorithm 2 to be $\epsilon/2$, let $\epsilon_{\mathcal{F}}$ satisfy the assumptions in Theorem 2, and let $\epsilon_0 = 0$. Thus the right hand side in Theorem 3 can be becomes ϵ and we proved Theorem 2.

B.1 Accuracy of action classification

First we define some notation that will be used in the proofs for TASID (Algorithm 2). Let the uniform distribution over action space be denoted $\texttt{Unf}(\mathcal{A})$. We will use $\texttt{Unf}(a) = 1/|\mathcal{A}|$ to denote the probability over action a given by uniform distribution. We define $P_h(x)$ to be the distribution from which x_h is sampled from in Line 9 in the hth iteration of Algorithm 2. Formally, $P_h(x_h)$ is the probability of receiving observation x_h at time step h when sampling the first $h-1$ actions according to the learned practicable policy $\pi_{1:h-1}$ in the target environment (when $h = 1$, then x_1 is directly emitted from the start state, i.e., $x_1 \sim q(\cdot \mid s_{\text{init}})$). Finally, we denote the joint distribution $P_h(x, a, x') := P_h(x) \times \texttt{Unf}(\mathcal{A}) \times T^\star_h(x'|x, a)$ from which (x_h, a_h, x_{h+1}) are sampled from in Line 9.

Now we show the form of conditional distribution of action given two observations, $f^\star_h(a \mid x, x')$, under the joint distribution $P_h(x, a, x')$. Moreover, we prove the form of $f^\star_h(a \mid x, x')$ for any roll-in distribution of the first observation x is the same.

Lemma 4. *Given any joint distribution $P(x, a, x') = P(x)\texttt{Unf}(a)T^\star_h(x' \mid x, a)$, where $P(x)$ is any prior distribution over x and $\texttt{Unf}(a)$ denotes the uniform distribution over actions, the posterior distribution over action a given pair of observation x, x' is:*

$$f^\star_h(a \mid x, x') = \frac{T^\star_h(x'|x, a)}{\sum_{a'} T^\star_h(x'|x, a')} = \frac{T^\star_h(\phi^\star(x')|\phi^\star(x), a)}{\sum_{a'} T^\star_h(\phi^\star(x')|\phi^\star(x), a')} \tag{33}$$

Proof. We start by applying the Bayes' rule.

$$f^\star_h(a \mid x, x') = \frac{P(x, a, x')}{\sum_{a' \in \mathcal{A}} P(x, a', x')} = \frac{P(x)\texttt{Unf}(a)T^\star_h(x'|x, a)}{\sum_{a' \in \mathcal{A}} P(x)\texttt{Unf}(a')T^\star_h(x'|x, a')} \tag{34}$$

$$= \frac{P(x)/|\mathcal{A}|T^\star_h(x'|x, a)}{P(x)/|\mathcal{A}| \sum_{a' \in \mathcal{A}} T^\star_h(x'|x, a')} \tag{35}$$

$$= \frac{T^\star_h(x'|x, a)}{\sum_{a' \in \mathcal{A}} T^\star_h(x'|x, a')} \tag{36}$$

$$= \frac{T^\star_h(\phi^\star(x') \mid \phi^\star(x), a)}{\sum_{a' \in \mathcal{A}} T^\star_h(\phi^\star(x') \mid \phi^\star(x), a')}, \tag{37}$$

where the last equality uses $T^\star_h(x' \mid x, a') = q(x' \mid \phi^\star(x'))T^\star_h(\phi^\star(x') \mid \phi^\star(x), a')$ for all $a' \in \mathcal{A}$, which follows from the block MDP structure of $M^\star$. □

Note that, in particular, Lemma 4 holds for the joint distribution $P_h(x, a, x')$ that we described earlier.

Now we can show the error bound of empirical log-likelihood maximizer. Before that, we introduce some useful lemmas from Agarwal et al. [2020] for the completeness of this paper.

Lemma 5 (Lemma 24 in Agarwal et al. [2020])**.** *Let $\mathcal{D}$ be a dataset of n (x, a, x') samples, and let $\mathcal{D}'$ be another independent dataset from the same distribution. Let $L(f, \mathcal{D}) = \sum_{i=1}^n \ell(f, (x_i, y_i))$ be any function that decomposes additively across examples where ℓ is any function, and let $\hat{f}(\mathcal{D})$ be any estimator taking as input random variable $\mathcal{D}$ and with range $\mathcal{F}$. Then, with probability 1- δ,*

$$-\log \mathrm{E}_{\mathcal{D}'} \exp(L(\hat{f}(\mathcal{D}), \mathcal{D}')) \leq -L(\hat{f}(\mathcal{D}), \mathcal{D}) + \log |\mathcal{F}| + \log(1/\delta)$$

Lemma 6 (Lemma 25 in Agarwal et al. [2020])**.** *For any two conditional probability densities f_1, f_2 and any distribution $D \in \Delta(\mathcal{X} \times \mathcal{X})$ we have*

$$\mathrm{E}_{x,x' \sim D} \|f_1(\cdot|x, x') - f_2(\cdot|x, x')\|^2_{TV} \leq -2 \log \mathrm{E}_{x,x' \sim D, a \sim f_2(\cdot|x,x')} \exp\left(-\frac{1}{2} \log \frac{f_2(a|x, x')}{f_1(a|x, x')}\right)$$

The proof of the empirical log-likelihood maximizer's error bound follows the proof in Agarwal et al. [2020] as well. However we modified the proof to only assume approximate realizability assumption instead of perfect realizaibility.

Theorem 4. *Let $\mathcal{D}_h$ be a data set with n_D transitions (x,a,x') sampled i.i.d. from $P_h(x, a, x')$. Let $f_h(a \mid x, x')$ and $\widetilde{f}_h(a \mid x, x')$ denote the maximizers of empirical log-likelihood and expected log-likelihood respectively:*

$$\widetilde{f}_h(a \mid x, x') \coloneqq \operatorname*{argmax}_{f \in \mathcal{F}} \mathrm{E}_{x,a,x' \sim P_h}[\ln f(a|x, x')] \tag{38}$$

$$f_h(a \mid x, x') \coloneqq \operatorname*{argmax}_{f \in \mathcal{F}} \sum_{(x_h, a_h, x_{h+1}) \in \mathcal{D}_h} \ln f(a_h \mid x_h, x_{h+1}) \tag{39}$$

Then for any $\delta \in (0, 1)$, with probability at least $1 - \delta$ we have that:

$$\mathrm{E}_{x,x' \sim P_h}\left[\|f_h(\cdot|x, x') - f_h^\star(\cdot|x, x')\|_{TV}^2\right] \leq \tfrac{2\log(|\mathcal{F}|/\delta)}{n_D} + \epsilon_{\mathcal{F}} \tag{40}$$

Proof. We set $L(f, \mathcal{D})$ in Lemma 5 to be $\sum_{(x,a,x') \in \mathcal{D}} -\frac{1}{2} \log \frac{\widetilde{f}_h(a|x,x')}{f_h(a|x,x')}$ where $\mathcal{D}$ is $\mathcal{D}_h$ and $\mathcal{D}'$ to be another I.I.D. dataset with n_D samples drawn independently with $\mathcal{D}$. With this choice, the right hand side is

$$\sum_{(x,a,x') \in \mathcal{D}} \frac{1}{2} \log \frac{\widetilde{f}_h(a|x, x')}{f_h(a|x, x')} + \log|\mathcal{F}| + \log(1/\delta) \leq \log|\mathcal{F}| + \log(1/\delta) \tag{41}$$

since f_h is the empirical maximum likelihood estimator and $f_h^\star \in \mathcal{F}$. The left hand side is

$$-\log \mathrm{E}_{\mathcal{D}'}\left[\exp\left(\sum_{(x,a,x') \in \mathcal{D}'} -\frac{1}{2} \log \frac{\widetilde{f}_h(a|x, x')}{f_h(a|x, x')}\right) \mid \mathcal{D}\right] \tag{42}$$

$$= -n_D \log \mathrm{E}_{x,a,x' \sim P_h} \exp\left(-\frac{1}{2} \log \frac{\widetilde{f}_h(a|x, x')}{f_h(a|x, x')}\right) \tag{43}$$

$$= -n_D \log \mathrm{E}_{P_h} \exp\left(-\frac{1}{2} \log \frac{f_h^\star(a|x, x')}{f_h(a|x, x')} - \frac{1}{2} \log \frac{\widetilde{f}_h(a|x, x')}{f_h^\star(a|x, x')}\right) \tag{44}$$

$$= -n_D \log \mathrm{E}_{P_h}\left[\exp\left(-\frac{1}{2} \log \frac{f_h^\star(a|x, x')}{f_h(a|x, x')}\right) \cdot \sqrt{\frac{f_h^\star(a|x, x')}{\widetilde{f}_h(a|x, x')}}\right] \tag{45}$$

By Assumption 3, we have that $\sqrt{\frac{f_h^\star(a|x,x')}{\widetilde{f}_h(a|x,x')}}$ is upper bounded by $\sqrt{1 + \epsilon_{\mathcal{F}}}$ uniformly overall x, a, x'. So we have that

$$-\log \mathrm{E}_{\mathcal{D}'}\left[\exp\left(\sum_{(x,a,x') \in \mathcal{D}'} -\frac{1}{2} \log \frac{\widetilde{f}_h(a|x, x')}{f_h(a|x, x')}\right) \mid \mathcal{D}\right] \tag{46}$$

$$\geq -n_D \log \mathrm{E}_{P_h}\left[\exp\left(-\frac{1}{2} \log \frac{f_h^\star(a|x, x')}{f_h(a|x, x')}\right)\right] - n_D \log \sqrt{1 + \epsilon_{\mathcal{F}}} \tag{47}$$

$$\geq \frac{n_D}{2} \mathrm{E}_{x,x' \sim P_h}\left[\|f_h(\cdot|x, x') - f_h^\star(\cdot|x, x')\|_{\mathrm{TV}}^2\right] - n_D \log \sqrt{1 + \epsilon_{\mathcal{F}}} \tag{48}$$

The last steps follows from Lemma 6. After combine this with the right hand side and rearrange the terms, we have

$$\mathrm{E}_{x,x' \sim P_h}\left[\|f_h(\cdot|x, x') - f_h^\star(\cdot|x, x')\|_{\mathrm{TV}}^2\right] \leq \frac{2\log(|\mathcal{F}|/\delta)}{n_D} + \log(1 + \epsilon_{\mathcal{F}}) \tag{49}$$

$$\leq \frac{2\log(|\mathcal{F}|/\delta)}{n_D} + \epsilon_{\mathcal{F}} \tag{50}$$

□

B.2 One-step accuracy of action decoder

In the last subsection, we have showed the learned function f_h approaches the posterior distribution of action at a rate of $1/n_D$. In Algorithm 2, the learned function f_h is used to build the state decoder α_h. This state decoding process relies on identified the correct "shadow" actions.

In this section we first show that there is a separation between the correct shadow action's probability and random actions' probabilities, in the posterior distribution of action. This relies on the transition dynamics in target environment lies in $\mathcal{C}(T^\circ, \eta)$. Then we show the action decoding accuracy. The idea to bound the 1-step action decoding accuracy is to apply the Markov's inequality on the event that the argmax action of f_h is wrong.

First, we define the set of shadow actions given two successive states.

Definition 3 (Shadow action set)**.** *For any states pair (s, s'), we define a set of actions $A_h(s, s') = \{a \in \mathcal{A} : T_h^\circ(s'|s, a) = 1\}$ and $A_h^c(s, s') = \mathcal{A} \backslash A_h(s, s')$.*

Since the observation emission function $q^\star$ maps different s to disjoint observation subspace, the shadow action sets can also be defined on the corresponding observation pairs x, x'. We define $A_h(x, x') = \{a \in \mathcal{A} : T_h^\circ(\phi^\star(x')|\phi^\star(x), a) = 1\}$, and similarly for $A_h^c(x, x')$

By definition, we have that $A_h(x, x') = A_h(s, s')$ if $x \sim q^\star(\cdot|s)$ and $x' \sim q^\star(\cdot|s')$. So later we may use these two notations interchangeably for convenience.

Next we prove the accuracy result of decoder α under the joint distribution with *any* practicable policy, but learned from the dataset from *uniform* action distribution. We use $\mathbb{1}$ as an indicator function of random events.

Lemma 7 (Accuracy of decoder)**.** *For any practicable policy π, let $P_{h,\pi}(x, a, x') := P_h(x)\pi(a|x)T^\star(x'|x, a)$ be the joint distribution of $P_h(x)$, policy π and transition function $T^\star$.*

Let $\mathcal{D}_h$ be a data set with n_D transitions (x,a,x') sampled i.i.d. from $P_h(x, a, x')$. For any $h \in [H]$ and $\delta \in (0, 1)$, and any practicable policy π, with probability at least $1 - \delta$, we have that

$$\mathrm{E}_{P_{h,\pi}}\left[\mathbb{1}(\alpha_h(x, x') \in A_h(x, x'))\right] \geq 1 - \frac{8h|\mathcal{A}|^3 \ln(|\mathcal{F}|/\delta)}{n_D(1 - 2\eta)^2} \tag{51}$$

Proof. For any $a_1 \in A_h(x, x')$, and $a_2 \in A_h^c(x, x')$,

$$f_h^\star(a_1 \mid x, x') - f_h^\star(a_2 \mid x, x') = \frac{T_h^\star(\phi^\star(x') \mid \phi^\star(x), a_1) - T_h^\star(\phi^\star(x') \mid \phi^\star(x), a_2)}{\sum_{a' \in \mathcal{A}} T_h^\star(\phi^\star(x') \mid \phi^\star(x), a')} \tag{52}$$

$$\geq \frac{(1 - \eta) - \eta}{\sum_{a' \in \mathcal{A}} T_h^\star(\phi^\star(x') \mid \phi^\star(x), a')} \geq \frac{(1 - 2\eta)}{|\mathcal{A}|} \tag{53}$$

Given that the gap is $(1-2\eta)/|\mathcal{A}|$, we have that for any fixed x, x', if $\|f_h(\cdot, x, x') - f_h^\star(\cdot, x, x')\|_{\mathrm{TV}} < (1-2\eta)/2|\mathcal{A}|$, then $\operatorname{argmax} f_h(\cdot, x, x') \in A_h(x, x')$.

Recall that $\alpha_h(x, x') = \operatorname{argmax}_{a \in \mathcal{A}} f_h(a \mid x, x')$. We have

$$\mathrm{E}_{P_h}[\mathbb{1}(\alpha_h(x, x') \notin A_h(x, x'))] = \Pr\left(\alpha_h(x, x') \notin A_h(x, x')\right) \tag{54}$$

$$\leq \Pr\left(\|f_h^\star(\cdot \mid x, x') - f_h(\cdot \mid x, x')\|_{\mathrm{TV}} \geq \frac{(1 - 2\eta)}{2|\mathcal{A}|}\right) \tag{55}$$

$$= \Pr\left(\|f_h^\star(\cdot \mid x, x') - f_h(\cdot \mid x, x')\|_{\mathrm{TV}}^2 \geq \frac{(1 - 2\eta)^2}{4|\mathcal{A}|^2}\right) \tag{56}$$

$$\leq \frac{\mathrm{E}\left[\|f_h^\star(\cdot \mid x, x') - f_h(\cdot, |x, x')\|_{\mathrm{TV}}^2\right]}{(1-2\eta)^2/4|\mathcal{A}|^2} \tag{57}$$

$$\leq \frac{8|\mathcal{A}|^2 \ln(|\mathcal{F}|/\delta)}{n_D(1 - 2\eta)^2} + \frac{4|\mathcal{A}|^2 \epsilon_{\mathcal{F}}}{(1 - 2\eta)^2} \tag{58}$$

The second to last step follows from Markov's inequality, and the last step follows from Theorem 4.

Notice that this is the error under distribution of uniform action $P_h(x, a, x') := P(x) \circ \texttt{Unf} \circ T_h^\star$. For any practicable policy π, since $\frac{P_{h,\pi}(x,a,x')}{P_h(x,a,x')} = \frac{\pi(a|x)}{1/|\mathcal{A}|} \leq |\mathcal{A}|$,

$$\mathrm{E}_{P_{h,\pi}}\left[\mathbb{1}(\alpha_h(x, x') \notin A_h(x, x'))\right] \leq \frac{8|\mathcal{A}|^3 \ln(|\mathcal{F}|/\delta)}{n_D(1-2\eta)^2} + \frac{4|\mathcal{A}|^3\epsilon_{\mathcal{F}}}{(1-2\eta)^2} \tag{59}$$

Then taking the complement of the event in the indicator function finished the proof. □

B.3 Analysis of the sample complexity

Now we are going prove that the learned policy recover the input latent policy with high probability.

Lemma 8. *For any $h \in [H]$, let $\mathbf{x}_{1:h+1} := x_1, x_2, \ldots, x_{h+1}$ and $\mathbf{a}_{1:h} := a_1, a_2, \ldots, a_h$ be the state and action sequence generated from $(\pi_1, \ldots, \pi_h)$ output by Algorithm 2. Given the high probability event in Theorem 4, we have that for any $h \in [H]$*

$$\mathrm{E}_{\pi_{1:h}}\left[\sum_{k=1}^{h} \mathbb{1}\{a_k \neq \rho(\phi(x_k))\}\right] \leq \frac{8h^2|\mathcal{A}|^3 \ln(|\mathcal{F}|/\delta)}{n_D(1-2\eta)^2} + \frac{4h^2|\mathcal{A}|^3\epsilon_{\mathcal{F}}}{(1-2\eta)^2} + \epsilon_0 \tag{60}$$

Proof. If the action decoder α_k is correct, i.e. $\alpha_k(x_k, x_{k+1}) \in A_k(x_k, x_{k+1})$ for any $k \in [h]$. Then if the initial state is s_{init}, we have that for any $k \in [h]$, $\phi_k(\mathbf{x}_{1:k}) = \phi^\star(x_{k+1})$ due to the determinism of the abstract simulator. Then for any $k \in [h]$,

$$a_k = \pi_k(\mathbf{x}_{1:k}) = \rho(\phi_k(\mathbf{x}_{1:k})) = \rho(\phi^\star(x_k)) \tag{61}$$

That means if we have $a_k \neq \rho(\phi^\star(x_k))$, we must have $\alpha_k(x_k, x_{k+1}) \notin A_k(x_k, x_{k+1})$ for some $j \leq k$, or the initial state is not s_{init}. So under the condition that the initial state is s_{init},

$$\mathbb{1}\{a_k \neq \rho(\phi^\star(x_k))\} \leq \sum_{j=1}^{k} \mathbb{1}\{\alpha_j(x_j, x_{j+1}) \notin A_j(x_j, x_{j+1})\} \tag{62}$$

Thus we have

$$\begin{aligned}
&\mathrm{E}_{\pi_{1:h}}\left[\sum_{k=1}^{h} \mathbb{1}\{a_k \neq \rho(\phi^\star(x_k))\}\right] && (63)\\
=&\mathrm{E}_{\pi_{1:h}}\left[\sum_{k=1}^{h}\sum_{j=1}^{k} \mathbb{1}\{\alpha_j(x_j, x_{j+1}) \notin A_j(x_j, x_{j+1})\}\right] + \epsilon_0 && (64)\\
\leq&\mathrm{E}_{\pi_{1:h}}\left[h\sum_{j=1}^{h} \mathbb{1}\{\alpha_j(x_j, x_{j+1}) \notin A_j(x_j, x_{j+1})\}\right] + \epsilon_0 && (65)\\
\leq&\frac{8h^2|\mathcal{A}|^3 \ln(|\mathcal{F}|/\delta)}{n_D(1-2\eta)^2} + \frac{4h^2|\mathcal{A}|^3\epsilon_{\mathcal{F}}}{(1-2\eta)^2} + \epsilon_0 && \text{(Lemma 7)}
\end{aligned}$$

□

This immediately gives the following theorem by letting $h = H$ and bounding the value gap for non-optimal actions by H. The proof of Theorem 2 follows from this.

Proof. We first bound the value gap using preivous lemma and the Performance Difference Lemma [Kakade, 2003].

$$
\begin{aligned}
V^{\pi}_{\mathbf{M}^\star} - v^{\rho\circ\phi^\star}_{\mathbf{M}^\star} \leq & \mathrm{E}_{x_h,a_h\sim\pi}\left[\sum_{h=1}^{H} Q^{\rho\circ\phi^\star}_{\mathbf{M}^\star,h}(x_h,a_h) - V^{\rho\circ\phi^\star}_{\mathbf{M}^\star,h}(x_h)\right] && \text{(Performance Difference)}\\
= & \mathrm{E}_{s_h,a_h\sim\pi}\left[\sum_{h=1}^{H} Q^{\rho}_{M^\star,h}(s_h,a_h) - V^{\rho}_{M^\star,h}(s_h)\right] && \text{(Block MDP)}\\
\leq & \mathrm{E}_{s_h,a_h\sim\pi}\left[\sum_{h=1}^{H} H\,\mathbb{1}\{a_h \neq \rho(\phi(x_h))\}\right] && \text{(Any } s_h, a_h, Q^{\rho}_h(s_h,a_h) \in [0,H])\\
\leq & \frac{8H^3|\mathcal{A}|^3\ln(|\mathcal{F}|/\delta)}{n_D(1-2\eta)^2} + H\epsilon_0 && \text{(Lemma 8 for } h = H)\\
= & \frac{8H^4|\mathcal{A}|^3\ln(|\mathcal{F}|/\delta)}{n(1-2\eta)^2} + \frac{4H^4|\mathcal{A}|^3\epsilon_{\mathcal{F}}}{(1-2\eta)^2} + H\epsilon_0 && (n = Hn_D)
\end{aligned}
$$

Finally, we can solve the sample complexity by denote the value gap ϵ, and finish the proof. □

C Analysis with Stochastic Initial State

We can extend TASID to a more general setting where initial state can be stochastically chosen, instead of being deterministic. This setting captures problems such as navigation in a set of house simulators, where dynamics of each house simulator can be deterministic but the choice of initial state, i.e., choice of current house and position of the agent inside the house, can be stochastically chosen.

As Algorithm 1 does not rely on the deterministic initial state, therefore, we only need to show that we can extend Algorithm 2 to the stochastic initial state setting, and find the robust policy learned by Algorithm 1.

First, we introduce the difference in problem settings and our main results under this setting formally. We assume that in both abstract simulator M° and the target environment $\mathbf{M}^\star$, the initial states are sampled from the same distribution μ over a finite set of initial states $\mathcal{S}_1$ of size $|\mathcal{S}_1| = N$.

We make an assumption that each initial state occurs with a reasonable probability and has a different transition dynamics.

Assumption 4. *(Conditions on initial state) For all initial states $s \in \mathcal{S}_1$, we assume $\mu(s) \geq \mu_{min}$ for some $\mu_{min} \in (0,1]$. Further, there exists a margin $\Gamma > 0$ such that for any two initial states $s, \tilde{s} \in \mathcal{S}_1$ we have:*

$$\|T(\cdot \mid s,a) - T(\cdot \mid \tilde{s},a)\|_{\mathtt{TV}} \geq \Gamma.$$

Informally, this assumption is required so that we can visit each initial state sufficiently, and use the margin assumption to learn an accurate decoder to cluster initial states. However, note that we cannot directly cluster in the observation space since we do not want to make any additional structural assumptions on it. In contrast, we will use a function-approximation approach where we interact with the observation via a function class.

We present a variation of TASID in Algorithm 3 that can address stochastic initial state. The algorithm assumes access to an approximate decoder $\hat{\phi} : \mathcal{X} \to \mathbb{N}$ that can cluster observations from the same initial state together. This decoder can be thought of partitioning the observation space into *decoder states* which recover the true initial states upto relabeling. In Section C.1 we discuss how to learn this decoder using the Homer algorithm Misra et al. [2020]. Algorithm 3 learns a mapping from these decoder states to initial states in $\mathcal{S}_1$ by performing a hypothesis testing algorithm that uses Algorithm 2 in the main paper as a sub-routine. In Section C.2 we discuss this hypothesis test.

We will prove that under a realizability assumption (stated later), this algorithm has the following guarantee.

Algorithm 3 TASID with multiple initial states

Input: An approximate clustering function of initial states $\hat{\phi}$, number of initial state N, error bound ϵ, failure probability δ.
Initialize counter $\texttt{step}(i) \leftarrow 0$ for $i \in [N]$
Initialize state map $\texttt{map}(i) \leftarrow -1$ for $i \in [N]$
for episode $e = 1, \ldots$ **do**
  For the initial observation x, decode the state by $\hat{\phi}(x)$.
  if $\texttt{map}(\hat{\phi}(x)) < 0$ **then**
    $\texttt{map}(\hat{\phi}(x)) \leftarrow$ InitialStateTest$(\hat{\phi}(x), \texttt{step}(\hat{\phi}(x)), N, \epsilon, \delta)$
    $\texttt{step}(\hat{\phi}(x)) \leftarrow \texttt{step}(\hat{\phi}(x)) + 1$
  else
    Run TASID on $\texttt{map}(\hat{\phi}(x))$

Algorithm 4 InitialStateTest

Input: state cluster i, global episode counter t, number of initial state N, error bound ϵ, failure probability δ.
if t = 0 **then**
  Hypothetic state $s \leftarrow 0$
  Initialize counter $\texttt{cnt}(i) \leftarrow 0$ for $i \in [N]$
  Initialize value estimates $v(i) \leftarrow 0$ for $i \in [N]$
  $n_l \leftarrow \frac{8H^4|\mathcal{A}|^3 \ln(N^2|\mathcal{F}|/\delta)}{\epsilon(1-2\eta)^2}$
  $n_t \leftarrow \frac{H^2 \ln(N/\delta)}{2\epsilon^2}$
  The algorithm instance will maintain the hypothesis state s, an episodes counter $\texttt{cnt}(\cdot)$ and a value logger $v(\cdot)$ for the same state cluster i across calls.
if $\texttt{cnt}(s) < n_l$ **then**
  Run TASID for one episode on initial state s
  $\texttt{cnt}(s) \leftarrow \texttt{cnt}(s) + 1$
  return -1
else if $n_l \geq \texttt{cnt}(s) < n_l + n_t$ **then**
  Rollout learned policy π for one episode and update value average $v(s)$
  $\texttt{cnt}(s) \leftarrow \texttt{cnt}(s) + 1$
  return -1
else
  $s \leftarrow s + 1$
  if $s = N + 1$ **then**
    Run TASID for one episode on initial state $\operatorname{argmax}_s v(s)$
    return $\operatorname{argmax}_s v(s)$
  else
    return -1

Theorem 5. *With the assumption about $\epsilon_{\mathcal{F}}$ in Theorem 2, Algorithm 3 will execute a policy that is close to robust policy by ϵ in all but poly $\left\{N, H, A, \frac{1}{\epsilon}, \frac{1}{(1-2\eta)}, \frac{1}{\mu_{min}} \ln\{\frac{1}{\delta}\}\right\}$ episodes with probability at least $1 - \delta$.*

Note that when there is a deterministic initial state, i.e., $N = 1$ and $\mu_{\text{min}} = 1$, we recover dependence on the same set of parameters as our main result. For some problems, N maybe significantly smaller than the set of all states. For example, a robot may start an episode from its charging station and there maybe a small number of charging stations in the environment. For these problems, the dependence on N may be acceptable.

In the second part, we propose a hypothesis testing based algorithm, that use Algorithm 2 in the main paper as a sub-routine, and prove the new sample complexity.

C.1 Learning decoder initial states

We use the Homer algorithm [Misra et al., 2020] to learn the decoder $\hat{\phi}$. We briefly describe the application of this algorithm for time step $h = 1$. Homer collects a dataset $\mathcal{D}$ of n quads as follows: we sample y uniformly in $\{0, 1\}$ and collect two independent transitions $(x^{(1)}, a^{(1)}, x'^{(1)}), (x^{(2)}, a^{(2)}, x'^{(2)})$ at the first time step by taking actions $a^{(1)}$ and $^{(2)}$ uniformly. If $y = 1$ then we add $(x^{(1)}, a^{(1)}, x'^{(1)}, y)$ to $\mathcal{D}$, otherwise, we add $(x^{(1)}, a^{(1)}, x'^{(2)}, y)$. Note that $(x^{(1)}, a^{(1)}, x'^{(2)})$ is an unobserved transition, therefore, we call it an imposter transition, whereas, $(x^{(1)}, a^{(1)}, x'^{(1)})$ is a real transition. We know that there are exactly N initial states since we have access to the simulator. Given a bottleneck function class $\Phi : \{\phi : \mathcal{X} \to [N]\}$ and another regressor class $\mathcal{G} : \{f : [N] \times \mathcal{A} \times \mathcal{X} \to [0, 1]\}$, we train a model to differentiate between real and imposter transition as follows:

$$\hat{g}, \hat{\phi} = \arg \min_{g \in \mathcal{G}, \phi \in \Phi} \sum_{(x,a,x',y) \in \mathcal{D}} \left(g(\phi(x), a, x') - y \right)^2 .$$

Difference from Misra et al. [2020]. While our approach and analysis in this subsection closely follows Misra et al. [2020], we differ from them in two crucial ways. Firstly, we apply bottleneck on x instead of x', since we want to recover a decoder for initial states. Secondly, Misra et al. [2020] do not assume a margin assumption Γ since their approach does not concern with recovering an exact decoder, but only in learning a good set of policies for exploration.

We will denote the function class $\mathcal{G} \circ \Phi = \{g \circ \phi : (x, a, x') \mapsto g(\phi(x), a, x') \mid g \in \mathcal{G}, \phi \in \Phi\}$. Let $D(x, a, x')$ be the marginal distribution over real and imposter transitions, and let $\rho(x') = \mathrm{E}_{x \sim \mu, a \sim \mathtt{unf}(\mathcal{A})} [T(x' \mid x, a)]$ be the marginal probability over x' for real transitions where μ is the initial state distribution. It can be shown that:

$$D(x, a, x') = \frac{\mu(x)}{2|\mathcal{A}|} \left\{ T(x' \mid x, a) + \rho(x') \right\}, \tag{66}$$

where $T(x'|x,a)\mu(x)/|\mathcal{A}|$ is the probability of observing a real transition (x, a, x') and $\rho(x')\mu(x)/|\mathcal{A}|$ is the probability of observing an imposter transition (x, a, x') and the factor of $1/2$ comes due to uniform selection over real and imposter transition.

Misra et al. [2020] showed that the Bayes optimal classifier of the prediction problem is given by:

Lemma 9 (Bayes Optimal Classifier)**.** *For any (x, a, x') in support of D, we have:*

$$g^\star(x, a, x') = \frac{T(x' \mid x, a)}{T(x' \mid x, a) + \rho(x')} = \frac{T(\phi^\star(x') \mid \phi^\star(x), a)}{T(\phi^\star(x') \mid \phi^\star(x), a) + \rho(\phi^\star(x'))}$$

Proof. See Lemma 9 of Misra et al. [2020]. □

Similar to Misra et al. [2020], we make a realizability assumption stated below that allows us to solve the classification problem well.

Assumption 5 (Realizability)**.** *We assume that $g^\star \in \mathcal{G} \circ \Phi$.*

We can use the realizability assumption to get the following generalization bound guarantee: for any $\delta \in (0, 1)$ we have:

$$\mathrm{E}_{x,a,x' \sim D} \left[\left| \hat{g}(\hat{\phi}(x), a, x') - g^\star(x, a, x') \right| \right] \leq \Delta := \sqrt{\frac{C(\mathcal{G} \circ \Phi)}{n} \ln \left(\frac{1}{\delta} \right)}, \tag{67}$$

with probability at least $1 - \delta$, where $C(\mathcal{G} \circ \Phi)$ is a complexity measure for class $\mathcal{G} \circ \Phi$ such as $\ln(|\mathcal{G}||\Phi|)$ or Rademacher complexity. For proof see Proposition 11 and Corollary 6 in Misra et al. [2020]. Note that even though their proof uses a bottleneck model ϕ on x' instead of x, essentially the same argument holds by symmetry.

Coupling Distribution. We define a coupling distribution as

$$D_c(x_1, x_2, a, x') = D(x_1 \mid a, x') D(x_2 \mid a, x') \frac{1}{|\mathcal{A}|} \rho(x'), \tag{68}$$

where $D(x_1 \mid a, x')$ is the conditional distribution derived from the joint distribution $D(x_1, a, x')$ defined earlier. We also define the marginal distribution $D(a, x')$ which gives us $D(x_1 \mid a, x') = {}^{D(x_1,a,x')}/_{D(a,x')}$.

We present some result related to the distributions defined above that will be useful later for proving important results later.

$$D(a, x') = \sum_x D(x, a, x') = \sum_x \frac{\mu(x)}{2|\mathcal{A}|}\{T(x' \mid x, a) + \rho(x')\} \geq \frac{\rho(x')}{2|\mathcal{A}|}, \tag{69}$$

$$\sum_a D(a, x') = \rho(x'). \tag{70}$$

Using Equation 69 we can prove:

$$\sum_{x_1} D_c(x_1, x_2, a, x') = \sum_{x_1} D(x_1 \mid a, x') \frac{D(x_2, a, x')}{D(a, x')} \frac{\rho(x')}{|\mathcal{A}|} \tag{71}$$

$$\leq 2 \sum_{x_1} D(x_1 \mid a, x') D(x_2, a, x') = 2D(x_2, a, x'). \tag{72}$$

Similarly, we can prove:

$$\sum_{x_2} D_c(x_1, x_2, a, x') \leq 2D(x_1, a, x'). \tag{73}$$

Further, we have:

$$D(x \mid a, x') = \frac{\mu(x)\{T(x' \mid x, a) + \rho(x')\}}{2|\mathcal{A}|D(a, x')} \geq \frac{\mu(x)\rho(x')}{2|\mathcal{A}|D(a, x')} \geq \frac{\mu(x)\rho(x')}{2|\mathcal{A}| \sum_{a \in \mathcal{A}} D(a, x')} = \frac{\mu(x)}{2|\mathcal{A}|},$$

which gives us:

$$\frac{D(x \mid a, x')\rho(x')}{T(x' \mid x, a) + \rho(x')} = \frac{\mu(x)\rho(x')}{2|\mathcal{A}|D(a, x')} \geq \frac{\mu(x)}{2|\mathcal{A}|}. \tag{74}$$

We now state a useful lemma.

Lemma 10. *Fix $\delta \in (0, 1)$. Then with probability at least $1 - \delta$ we have*

$$\mathrm{E}_{x_1, x_2, a, x' \sim D_c} \left[\mathbf{1}\{\hat{\phi}(x_1) = \hat{\phi}(x_2)\} |g^\star(x_1, a, x') - g^\star(x_2, a, x')| \right] \leq 4\Delta.$$

Proof. We use triangle inequality to decompose the left hand side as:

$$\begin{aligned}
&\mathrm{E}_{(x_1, x_2, a, x') \sim D_c} \left[\mathbf{1}\{\hat{\phi}(x_1) = \hat{\phi}(x_2)\} |g^\star(x_1, a, x') - g^\star(x_2, a, x')| \right] \\
&\leq \mathrm{E}_{(x_1, x_2, a, x') \sim D_c} \left[\mathbf{1}\{\hat{\phi}(x_1) = \hat{\phi}(x_2)\} |g^\star(x_1, a, x') - \hat{g}(\hat{\phi}(x_1), a, x')| \right] + \\
&\mathrm{E}_{(x_1, x_2, a, x') \sim D_c} \left[\mathbf{1}\{\hat{\phi}(x_1) = \hat{\phi}(x_2)\} |\hat{g}(\hat{\phi}(x_1), a, x') - g^\star(x_2, a, x')| \right]
\end{aligned}$$

The first term is bounded as shown below:

$$\begin{aligned}
&\mathrm{E}_{(x_1, x_2, a, x') \sim D_c} \left[\mathbf{1}\{\hat{\phi}(x_1) = \hat{\phi}(x_2)\} |g^\star(x_1, a, x') - \hat{g}(\hat{\phi}(x_1), a, x')| \right] \\
&\leq \mathrm{E}_{(x_1, x_2, a, x') \sim D_c} \left[|g^\star(x_1, a, x') - \hat{g}(\hat{\phi}(x_1), a, x')| \right] \\
&\leq 2\mathrm{E}_{(x, a, x') \sim D} \left[|g^\star(x, a, x') - \hat{g}(\hat{\phi}(x), a, x')| \right] = 2\Delta,
\end{aligned}$$

where the second inequality uses Equation 73 and Equation 67. The second term is bounded as:

$$\begin{aligned}
&\mathrm{E}_{(x_1, x_2, a, x') \sim D_c} \left[\mathbf{1}\{\hat{\phi}(x_1) = \hat{\phi}(x_2)\} |\hat{g}(\hat{\phi}(x_1), a, x') - g^\star(x_2, a, x')| \right] \\
&= \mathrm{E}_{(x_1, x_2, a, x') \sim D_c} \left[\mathbf{1}\{\hat{\phi}(x_1) = \hat{\phi}(x_2)\} |\hat{g}(\hat{\phi}(x_2), a, x') - g^\star(x_2, a, x')| \right] \leq 2\Delta,
\end{aligned}$$

where the inequality results from following similar steps used for bounding the first term. Adding the two upper bounds we get 4Δ. □

Using Lemma 9, we have for every $x_1, x_2, x' \in \mathcal{X}, a \in \mathcal{A}$:

$$|g^\star(x_1, a, x') - g^\star(x_2, a, x')| = \frac{\rho(x')|T(x' \mid x_1, a) - T(x' \mid x_2, a)|}{(T(x' \mid x_1, a) + \rho(x'))(T(x' \mid x_2, a) + \rho(x'))}. \tag{75}$$

We use this to prove the following result:

Lemma 11. *With probability at least $1 - \delta$ we have:*

$$\Pr_{x_1, x_2 \sim \mu}\Big(\phi^\star(x_1) \neq \phi^\star(x_2) \wedge \hat{\phi}(x_1) = \hat{\phi}(x_2)\Big) \leq \frac{8|\mathcal{A}^2|\Delta}{\Gamma}.$$

Proof. Let's define a shorthand $\mathcal{E} = \mathbf{1}\{\hat{\phi}(x_1) = \hat{\phi}(x_2)\}$. Starting with left hand side of Lemma 10 we get:

$$\begin{aligned}
&\mathrm{E}_{(x_1,x_2,a,x') \sim D_c}\left[\mathcal{E}|g^\star(x_1, a, x') - g^\star(x_2, a, x')|\right] \\
&= \mathrm{E}_{(x_1,x_2,a,x') \sim D_c}\left[\mathcal{E}\frac{\rho(x')|T(x' \mid x_1, a) - T(x' \mid x_2, a)|}{(T(x' \mid x_1, a) + \rho(x'))(T(x' \mid x_2, a) + \rho(x'))}\right] \\
&= \sum_{x_1,x_2,a,x'} \frac{\mathcal{E}}{|\mathcal{A}|}\frac{D(x_1 \mid x', a)\rho(x')}{(T(x' \mid x_1, a) + \rho(x'))}\frac{D(x_2 \mid a, x')\rho(x')}{(T(x' \mid x_2, a) + \rho(x'))}|T(x' \mid x_1, a) - T(x' \mid x_2, a)| \\
&\geq \sum_{x_1,x_2,a,x'} \mathcal{E}\frac{\mu(x_1)\mu(x_2)}{4|\mathcal{A}|^3}|T(x' \mid x_1, a) - T(x' \mid x_2, a)|, \quad \text{using Equation 74} \\
&\geq \sum_{x_1,x_2,a} \mathbf{1}\{\phi^\star(x_1) \neq \phi^\star(x_2)\}\mathcal{E}\frac{\mu(x_1)\mu(x_2)}{2|\mathcal{A}|^3}\Gamma \\
&= \frac{\Gamma}{2|\mathcal{A}|^2}\Pr_{x_1,x_2 \sim \mu}\Big(\phi^\star(x_1) \neq \phi^\star(x_2) \wedge \hat{\phi}(x_1) = \hat{\phi}(x_2)\Big)
\end{aligned}$$

where the last inequality uses $\frac{1}{2}\sum_{x'} |T(x' \mid x_1, a) - T(x' \mid x_2, a)| = \|T(\cdot \mid x_1, a) - T(\cdot \mid x_2, a)\|_{\mathtt{TV}}$ which is either zero when $\phi^\star(x_1) = \phi^\star(x_2)$ or at least Γ. Combining these two conditions we get a lower bound of $\mathbf{1}\{\phi^\star(x_1) \neq \phi^\star(x_2)\}\Gamma$ on TV distance. The proof is then completed with application of Lemma 10. □

Theorem 6. *(Initial State Clustering Result). Let $N > 1$ and let $\Delta < \frac{\mu_{min}^2 \Gamma}{32N^2|\mathcal{A}|^2}\left(1 - \left(1 - \frac{2\mu_{min}}{N}\right)^2\right)$. Then there exists a bijection mapping $\sigma : \mathcal{S}_1 \to [N]$ such that with probability at least $1 - \delta$:*

$$\begin{aligned}
\forall s \in \mathcal{S}_1, \quad &\Pr_{x \sim \mu}\Big(\phi^\star(x) = s \mid \hat{\phi}(x) = \sigma(s)\Big) \\
&> 1 - \frac{16N^2|\mathcal{A}|^2\Delta}{\Gamma\mu_{min}^2} = 1 - \frac{16N^2|\mathcal{A}|^2}{\Gamma\mu_{min}^2}\sqrt{\frac{C(\mathcal{F} \circ \Phi)\ln(1/\delta)}{n}}.
\end{aligned} \tag{76}$$

Proof. We will use $i \in [N]$ to denote a decoder state defined by $\{x \mid \hat{\phi}(x) = i, x \in \mathcal{X}_1\}$ and $s \in \mathcal{S}_1$ to denote a real state defined by $\{x \mid \phi^\star(x) = s, x \in \mathcal{X}_1\}$. For any i, s we can bound the left hand side of Lemma 11 as:

$$\Pr(\phi^\star(x_1) \neq \phi^\star(x_2) \wedge \hat{\phi}(x_1) = \hat{\phi}(x_2)) \tag{77}$$

$$= \Pr(\cup_{j \in [N], \tilde{s} \in \mathcal{S}_1} \phi^\star(x_1) = \tilde{s}, \phi^\star(x_2) \neq \tilde{s}, \hat{\phi}(x_1) = j, \hat{\phi}(x_2) = j) \tag{78}$$

$$\geq \Pr(\phi^\star(x_1) = s, \phi^\star(x_2) \neq s, \hat{\phi}(x_1) = i, \hat{\phi}(x_2) = i) \tag{79}$$

$$= \Pr(\phi^\star(x_1) = s, \hat{\phi}(x_1) = i)\Pr(\phi^\star(x_2) \neq s, \hat{\phi}(x_2) = i) \tag{80}$$

$$= \Pr(\phi^\star(x) = s, \hat{\phi}(x) = i)\Big\{\Pr(\hat{\phi}(x) = i) - \Pr(\phi^\star(x) = s, \hat{\phi}(x) = i)\Big\} \tag{81}$$

where the second last step follows from observing that x_1 and x_2 are sampled independently. We define a few shorthands: $\Pr(i) = \Pr(\hat{\phi}(x) = i)$, $\Pr(s) = \Pr(\phi^\star(x) = s) = \mu(s)$ and $\Pr(i, s) = \Pr(\phi^\star(x) = s, \hat{\phi}(x) = i)$. This combined with above and Lemma 11 gives us:

$$\forall i \in [N], s \in \mathcal{S}_1, \qquad \Pr(i, s)\left(\Pr(i) - \Pr(i, s)\right) \leq \Delta' := \frac{8|\mathcal{A}|^2\Delta}{\Gamma} \tag{82}$$

We define a mapping $\sigma : \mathcal{S}_1 \to [N]$ as follows:

$$\sigma(s) = \arg\max_{i \in [N]} \Pr(i, s) \tag{83}$$

This gives us:

$$\Pr(\sigma(s), s)\left(\Pr(\sigma(s)) - \Pr(\sigma(s), s)\right) \leq \Delta' \tag{84}$$

Since Δ' can be brought arbitrarily small, we will assume $\Delta' < {\Pr(\sigma(s))^2}/{4}$ which allows us to write:

$$\Pr(\sigma(s), s) > \frac{\Pr(\sigma(s)) + \sqrt{\Pr(\sigma(s))^2 - 4\Delta'}}{2}, \text{ or} \tag{85}$$

$$\Pr(\sigma(s), s) < \frac{\Pr(\sigma(s)) - \sqrt{\Pr(\sigma(s))^2 - 4\Delta'}}{2} \tag{86}$$

By definition of $\sigma(s)$ we have:

$$\Pr(\sigma(s), s) \geq \frac{1}{N}\sum_{i=1}^{N} \Pr(i, s) = \frac{\Pr(s)}{N} \geq \frac{\mu_{\text{min}}}{N}, \tag{87}$$

where the first inequality uses the fact that maximum of a set of values is greater than its average, and the last inequality uses Assumption 4. We now place another condition on Δ', namely,

$$\Delta' < \frac{\Pr(\sigma(s))^2}{4}\left(1 - \left(1 - \frac{2\mu_{\text{min}}}{\Pr(\sigma(s))N}\right)^2\right), \tag{88}$$

which implies that:

$$\Pr(\sigma(s), s) < \frac{\Pr(\sigma(s)) - \sqrt{\Pr(\sigma(s))^2 - 4\Delta'}}{2} < \frac{\mu_{\text{min}}}{N}. \tag{89}$$

This eliminates Equation 86. Hence the $\Pr(\sigma(s), s)$ must satisfy Equation 85 which can be simplified as:

$$\begin{aligned}
\Pr(\sigma(s), s) &> \frac{\Pr(\sigma(s)) + \sqrt{\Pr(\sigma(s))^2 - 4\Delta'}}{2} && (90)\\
&= \frac{\Pr(\sigma(s))}{2}\left(1 + \left(1 - \frac{4\Delta'}{\Pr(\sigma(s))^2}\right)^{\frac{1}{2}}\right) && (91)\\
&\geq \Pr(\sigma(s))\left(1 - \frac{2\Delta'}{\Pr(\sigma(s))^2}\right) && (92)\\
&\geq \Pr(\sigma(s))\left(1 - \frac{2N^2\Delta'}{\mu_{\text{min}}^2}\right) && (93)
\end{aligned}$$

where the third step uses $\sqrt{1-y} \geq 1 - y$ for $y \in [0, 1]$ and that ${4\Delta'}/{\Pr(\sigma(s))^2} < 1$ from constraints on Δ', and the last step uses $\Pr(\sigma(s)) \geq \Pr(\sigma(s), s) \geq {\mu_{\text{min}}}/{N}$ (Equation 87). We can finally prove our main result as:

$$\Pr(s \mid \sigma(s)) = \frac{\Pr(s, \sigma(s))}{\Pr(\sigma(s))} \geq 1 - \frac{2N^2\Delta'}{\mu_{\text{min}}^2} = 1 - \frac{16N^2|\mathcal{A}|^2\Delta}{\Gamma\mu_{\text{min}}^2}.$$

What is left is to show that $\sigma(s)$ is a bijection mapping and collect all constraints on Δ'. Let s and s' be two initial states such that $\sigma(s) = \sigma(s') = k$. We then get:

$$\Pr(k, s)\Pr(k, s') \leq \Pr(k, s)\left(\Pr(k) - \Pr(k, s)\right) \leq \Delta' \tag{94}$$

where the last equality follows from Equation 84. Further, we have $\Pr(k, s) \geq {\mu_{\text{min}}}/{N}$ and $\Pr(k, s') \geq {\mu_{\text{min}}}/{N}$ from Equation 87. This gives us $\frac{\mu_{\text{min}}^2}{N^2} \leq \Delta'$. Hence, if $\Delta' < \frac{\mu_{\text{min}}^2}{N^2}$, then, we cannot have two different initial states mapping to the same decoder state. Further, as $|\mathcal{S}_1| = N$, hence the map $\sigma : \mathcal{S}_1 \to [N]$ is a bijection.

Finally, we made three constraints on Δ'. The first is $\Delta' < {\Pr(\sigma(s))^2}/{4}$, second is Equation 88, and third is $\Delta' < \frac{\mu_{\min}^2}{N^2}$. Note that Equation 88 already implies that $\Delta' < {\Pr(\sigma(s))^2}/{4}$. Hence, the overall constraint on Δ' is:

$$\Delta' < \min\left\{\frac{\mu_{\min}^2}{N^2}, \frac{\Pr(\sigma(s))^2}{4}\left(1-\left(1-\frac{2\mu_{\min}}{\Pr(\sigma(s))N}\right)^2\right)\right\} \tag{95}$$

We can simplify this constraint by making it tighter using $\Pr(\sigma(s)) \in \left[\frac{\mu_{\min}}{N}, 1\right]$:

$$\Delta' < \min\left\{\frac{\mu_{\min}^2}{N^2}, \frac{\mu_{\min}^2}{4N^2}\left(1-\left(1-\frac{2\mu_{\min}}{N}\right)^2\right)\right\} = \frac{\mu_{\min}^2}{4N^2}\left(1-\left(1-\frac{2\mu_{\min}}{N}\right)^2\right). \tag{96}$$

Note that we are assuming here that $N \geq 2$ and, therefore, $\mu_{\min} < 1$, which implies ${2\mu_{\min}}/{N} \leq \mu_{\min} < 1$. When $N = 1$, we can trivially align the single initial state. This completes the proof. □

This allows us to separate all initial states at time step $h = 1$ with high probability.

C.2 Aligning the learned decoder states

The only thing left is aligning learned decoder states in the target domain with simulator initial states, which we do as follows.

At a high level Algorithm 3 first clusters the initial observation into clusters, then each time it start with a cluster, it run a subroutine, InitialStateTest, to test the latent state index of that cluster. The testing algorithm Algorithm 4, run our main algorithm `TASID` the hypothesis of the state index from 0 to N. By the analysis of `TASID`, we know that if the hypothesis is correct, it will learn a policy with nearly robust value. Thus for each cluster, with in Nn episodes the algorithm will find the nearly robust policy.

Theorem 7. *If $\epsilon_{\mathcal{F}} \leq \frac{(1-2\eta)^2\epsilon}{4H^4|\mathcal{A}|^3}$ we learn the initial state decoder $\hat{\phi}$ with n_0 samples such that*

$$n_0 \geq \max\left\{\frac{256H^2N^4|\mathcal{A}|^4C(\mathcal{F}\circ\Phi)\ln(1/\delta)}{\epsilon^2\Gamma^2\eta^4}, \frac{1024N^4|\mathcal{A}|^4C(\mathcal{F}\circ\Phi)\ln(1/\delta)}{\Gamma^2\eta^6}\right\},$$

Algorithm 3 will execute a policy that is close to robust policy by 5ϵ in all but

$$\mathcal{O}\left(\frac{N^2H^4|\mathcal{A}|^3\ln(N^2|\mathcal{F}|/\delta)}{\epsilon(1-2\eta)^2} + \frac{N^2H^2\ln(N^2/\delta)}{\epsilon^2} + \frac{N^4|\mathcal{A}|^4C(\mathcal{F}\circ\Phi)\ln(1/\delta)}{\Gamma^2\eta^2}\max\left\{\frac{H}{\epsilon^2}, \frac{1}{\eta^2}\right\}\right)$$

episodes with probability at least $1-3\delta$.

Proof. We prove this theorem by two steps. First, we show that for each cluster $\hat{\phi}(x)$, after we run Algorithm 4 with $N(n_l+n_t)$ steps we can find the correct state of that cluster. Second, we show that once we find the correct state, we run `TASID` and learn a policy at most 4ϵ worse than the robust policy.

First, by the definition of n_0, we have that $\Delta := \sqrt{\frac{C(\mathcal{F}\circ\Phi)}{n_0}\ln\left(\frac{1}{\delta}\right)} \leq \frac{\eta^2\Gamma}{32N^2|\mathcal{A}|^2}\frac{2\eta}{N} < \frac{\eta^2\Gamma}{32N^2|\mathcal{A}|^2}\left(1-\left(1-\frac{2\eta}{N}\right)^2\right)$ for $N \geq 2$ and $\eta < 1$. By Theorem 6, we have for each $i \in [N]$, there must exist a state $\sigma(i) \in [N]$ such that with probability at least $1-\delta$,

$$\Pr(\phi(x) = s|\hat{\phi}(x) = i) \geq 1 - \frac{16N^2|\mathcal{A}|^2\Delta}{\Gamma\eta^2} \tag{97}$$

$$\geq 1 - \frac{\epsilon}{H} \tag{98}$$

Thus if we run Algorithm 4 InitialStateTest for $s = \sigma(i)$. Then by Theorem 3, we have that the value of learned policy π is at least

$$V_{\mathbf{M}}^{\rho\circ\phi^\star} - \frac{8H^4|\mathcal{A}|^3\ln(N^2|\mathcal{F}|/\delta)}{n_l(1-2\eta)^2} - \frac{4H^4|\mathcal{A}|^3\epsilon_{\mathcal{F}}}{(1-2\eta)^2} - H\epsilon_0 = V_{\mathbf{M}}^{\rho\circ\phi^\star} - 3\epsilon,$$

with probability at least $1 - \delta/N^2$. By Hoeffding's inequality, we know that the Monte-Carlo estimates $v(s)$ of the learned policy value with N_t samples is at least

$$V_{\mathbf{M}}^{\rho\circ\phi^\star} - 3\epsilon - \sqrt{\frac{H^2 \ln(N^2/\delta)}{2n_t}} = V_{\mathbf{M}}^{\rho\circ\phi^\star} - 4\epsilon,$$

with probability at least $1 - \delta/N^2$. Thus let $\hat{s} = \operatorname{argmax}_s v(s)$ and π be the corresponding learned policy from TASID given hypothetical initial state $\hat{s}$. The policy value v^π is at least

$$v(\hat{s}) - \epsilon \geq V_{\mathbf{M}}^{\rho\circ\phi^\star} - 5\epsilon \tag{99}$$

with probability $1 - 2\delta/N$ by taking the union bound on all N possible hypothetical initial states. Thus, for a given state cluster, after run Algorithm 4 InitialStateTest for $N(n_l + n_t)$ episodes, the policy value is at least $V_{\mathbf{M}}^{\rho\circ\phi^\star} - 5\epsilon$. That means for each state cluster, we make at most $N(n_l + n_t)$ mistakes. Thus we make at most $N^2(n_l + n_t)$ mistakes in total during running Algorithm 4. Notice that we also need n_0 samples to learn the decoder $\hat{\phi}$. The total number of episodes we may make mistake on is therefore

$$N^2(n_l + n_t) + n_0$$

Taking the union bound over all initial state clusters $\hat{\phi}(x)$ and the high probability statement in Theorem 6, we have that with probability $1 - 3\delta$ the statement holds. We finish the proof by plugging in n_l, n_t, n_0. □

D Experiment Details

D.1 Experiment Details in Combination Lock

Domain details We describe the details of the combination lock domain here. The deterministic MDP is described in section 5. The transition dynamics and reward functions in the target domain is defined by the the coefficients $\xi(a'|s,a)$ and Definition 1. For each s, a, we set $\xi(a'|s,a) := \eta p(a'|s) + (1-\eta)\,\mathbb{1}(a' = a)$ where $p(\cdot|s)$ is a random probability mass distribution and each probability is drawn uniformly from $[0,1]$ and then normalized. We set $\eta = 0.1$ in the experiment. The only exception is that all transitions from state $(H, 2)$ are not perturbed in the true target domain. This settings makes the robust policy without this knowledge not optimal in this specific instance, but our goal is still learning the robust policy here.

The observation mapping is defined as below. Let v_s be a 2-sparse encoding in $\mathbb{R}^{H+4}$ of the state (h, i) where the first 3 bits is a 1-sparse encoding of i and last $H+1$ bits is a 1-sparse encoding of h. The the observation o is computed by

$$o = \mathbf{H} \times \texttt{perm}(v_s + \mathcal{N}(0, 0.01)), \tag{100}$$

where perm is an arbitrary permutation of $[H+4]$, and $\mathbf{H}$ is the $2^{\lceil \log_2(H+4) \rceil}$ by $2^{\lceil \log_2(H+4) \rceil}$ Hadamard matrix, consists of $2^{\lceil \log_2(H+4) \rceil}$ mutually orthogonal columns. The permutation $\texttt{perm}(v)$ operator shuffles the dimensions of the vector v. E.g., if perm denotes the permutation $(3, 1, 2)$, then $\texttt{perm}(v) = (v_3, v_1, v_2)$. We create a Hadamard matrix H_n of size $n \times n$ for $n = 2^l$ for some $l \in \mathbb{N} \cup \{0\}$ using Sylvester's construction.[4] If $H + 4 < 2^{\lceil \log_2(H+4) \rceil}$, then the vector $\texttt{perm}(v_s + \mathcal{N}(0, 0.01))$ will be padded with zeros to ensure it is of size $2^{\lceil \log_2(H+4) \rceil}$. Our constructions are significantly based on previous hard combination locks studied in Du et al. [2019], Misra et al. [2020].

Implementation details of TASID We describe the details of TASID and hyper-parameters below. We model the action predictor class $\mathcal{F}$ by a two-layer MultiLayer Perceptron (MLP) with Leaky ReLU activations in this domain. The input to the MLP is the concatenation of observations x and x'. This is processed through two non-linear layers, and finally a softmax operation is performed to generate probabilities over actions. We implement the model in PyTorch and train it using Adam optimization.

[4] Sylvester's construction defines $H_1 = [1]$ and $H_{2k} = \begin{bmatrix} H_k & H_k \\ H_k & -H_k \end{bmatrix}$ for all $k \in \mathbb{N}$.

The $h+1$-th action predictor, i.e., $\hat{f}_{h+1}$ is initialized with the parameter from h-th action predictor (f_h). We initialize the first action predictor (f_1) using PyTorch's default initialization. For action predictor we remove 20% of the n_D datapoints for use as a held-out validation set and use the remaining for training parameters of the MLP. We perform 100 epochs of training and evaluate the current model on the validation set after each epoch. We stop training if the validation loss does not increase for 10 consecutive epochs. We save the model after each epoch, and use the model with the smallest validation loss. The only hyperparameter we search for TASID is n_D whose values we search over the grid $\{1000, 2500, 10000\}$.

For all algorithms we report results with best hyperparameter for every value of H. For every value of H and hyperparameter setting, we run the experiment 5 times with different seeds. We select the best hyperparameter as one that takes the least median number of episodes to achieve a moving average return of $0.95v^\rho$, where ρ is the latent robust policy and v^ρ is its value in the target domain. We compute median across 5 seeds. In case of a tie, we look at the average return over the entire course of training and across all seeds.

The hyper-parameters of training neural networks is the same across our methods and all baseline methods, and are specified in Table 2 on page 31.

parameter name	values
hidden dimension	56
learning rate	0.0003
optimizer	Adam
batch size	32
gradient clipping	0.25

Table 2: Hyperparameters in training NNs for all algorithms in the combination lock domain

Implementation details of PPO, PPO+RND, and domain randomization We train PPO and PPO+RND, with a maximum of 5×10^5 number of episodes in the target domain. The policy network is implemented by a two-layer MLP with ReLU activations in this domain, and the training hyperparameters are the same as Table 2. When training RND, the observation is normalized to mean 0 and standard deviation 1. Hyperparameters that are specific to the PPO and RND are listed in Table 3.

We perform a grid search over the entropy coefficient in PPO and the coefficient of RND bonus. The grid search we use for PPO+RND experiments contains 15 different hyperparameter choices (5 for RND bonus and 3 for entropy coefficient). This results in $15 \times 5 = 75$ experiments for every value of H. In contrast, the grid search for TASID only contains 3 different hyperparameter choices. To avoid further increasing the search space, we only use the best PPO+RND hyperparameters for every value of H when running the domain randomization experiments. The best hyperparameter values are chosen using the procedure described earlier with sole exception we use compute episodes needed to achieve a moving average return of $0.5v^\rho$ instead of $0.95v^\rho$. This weaker metric is more helpful since the latter often gives infinities.

For domain randomization pretraining, we first train in the source domain with 5×10^5 episodes. During this training, every 100 episodes, we randomize the domain uniformly by regenerating a permutation function and the perturbation in transitions $\xi(\cdot|\cdot,\cdot)$, in the same way as how the target domain is generated. We assume that the domain randomization algorithm knows η (same as TASID), and everything about the observation mapping except the specific permutation function and transition perturbation (not known by TASID).

Rather than uniform domain randomization, EPOpt [Rajeswaran et al., 2017] uses only episodes with a return smaller than lower ϵ-quantile in the batch (100 episodes) during domain randomization pretraining. We searched for values of ϵ in $\{0.1, 0.2, 0.5, 1.0\}$ where 1.0 gives uniform domain randomization. Therefore, the hyperparameter grid search for domain randomization has 4 different choices.

Details of the experiment results We report the details of experiments we used to generate Figure 2b in the paper. We run each algorithm with 5 random seeds for a maximum of 5×10^5 number

parameter name	values
clip epsilon	0.1
discounting factor	0.999
number of updates per batch	4
number of episodes per batch	100
entropy loss coefficient	$0, 0.01, 0.001$
RND bonus coefficient	$0, 100, 500, 1000, 10000$
ϵ in EPOpt	$0.1, 0.2, 0.5, 1.0$

Table 3: Hyperparameters for PPO, PPO+RND, PPO(+RND) with domain randomization in the combination lock domain. Multiple values stand for the values we run grid-search over, and we report the best performance among them for each environment specification (each H).

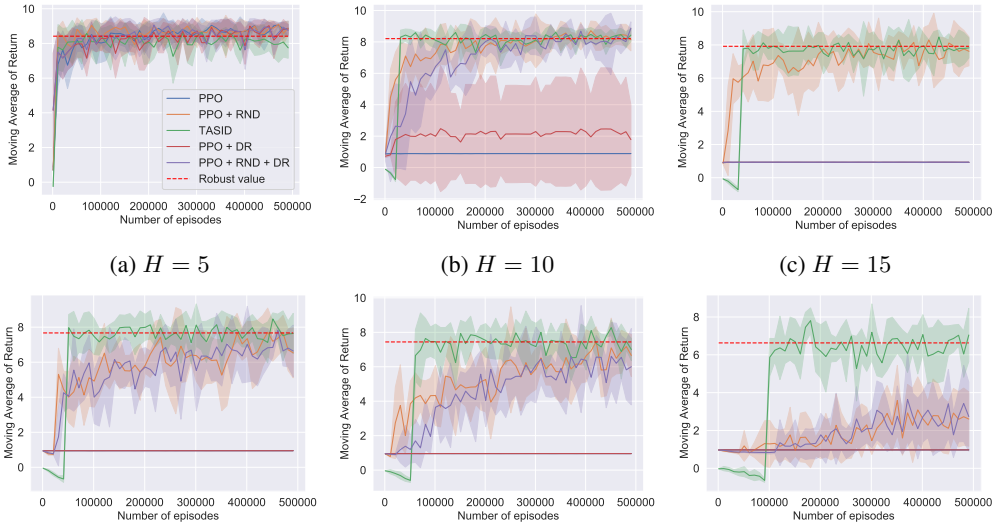
of episodes in the target domain, saving checkpoints every 1000 episodes. We decide whether the combination lock is solved or not if the algorithm can achieve a moving average return of $0.95v^\rho$. In Figure 2b we report the median of the number of episodes needed to solve the combination lock with horizon H. The results of each random seeds is list in the following table.

Algorithm	Horizon	Number of episodes needed
TASID	5	17000, 241000, 9000, 175000, 9000
	10	29000, 32000, 33000, 41000, 30000
	15	42000, 43000, 41000, 55000, 42000
	20	53000, 56000, 54000, 52000, 63000
	25	341000, 142000, 67000, 198000, 69000
	40	123000, 217000, 303000, 156000, 138000
PPO	5	69000, 128000, 58000, 31000, 81000
	≥ 10	∞, ∞, ∞, ∞, ∞
PPO+RND	5	41000, 18000, 12000, 17000, 27000
	10	141000, 152000, 103000, 245000, 106000
	15	102000, 318000, 167000, 147000, 111000
	20	459000, 384000, ∞, ∞, 349000
	25	∞, 383000, ∞, 458000, ∞
	40	∞, ∞, ∞, ∞, ∞
PPO+DR	5	91000, 77000, 53000, 186000, 66000
	≥ 10	∞, ∞, ∞, ∞, ∞
PPO+RND+DR	5	53000, 40000, 50000, 66000, 143000
	10	170000, 364000, 194000, 252000, 214000
	15	∞, ∞, ∞, ∞, ∞
	20	265000, ∞, ∞, ∞, ∞
	25	409000, ∞, ∞, 352000, ∞
	40	∞, ∞, ∞, ∞, ∞

Table 4: Results for all random seeds to generate Figure 2b. A value of ∞ denotes a timeout indicating that the algorithm was not able to achieve a moving average return of $0.95v^\rho$ in the maximum allowed number of episodes of 5×10^5.

In the main paper, Figure 2c shows the reward curves for different algorithms with horizon equals 40 in the target environment. Here we include the reward curves for all values of horizon in Figure 4. It shows our algorithm stably learned a robust policy with number of samples that is smaller than baseline algorithms. Though baseline algorithms aim to learn the near optimal policy, they did not converge to a policy that is significantly better than the robust policy.

Amount of compute We run our experiments on a cluster containing mixture of P40, P100, and V100 GPUs. Each experiment runs on a single GPU in a docker container. We use Python3 and



(a) $H = 5$ (b) $H = 10$ (c) $H = 15$

(d) $H = 20$ (e) $H = 25$ (f) $H = 40$

Figure 4: Reward curves in target environments with different horizons. The legend is shown in the plot for $H = 5$, and is shared with all other plots.

Pytorch 1.4. We found that for H=40 and V100 GPUs, on average, POTAS took 2.5 hours, PPO + DR took 14 hours, and PPO + RND + DR took 31 hours. Performing domain randomization increased the computational time by a factor of 2.

D.2 Experiment Details in MiniGrid

Domain details We use the code of the MiniGrid environment [Chevalier-Boisvert et al., 2018] under the Apache License 2.0. In MiniGrid, the agent is placed in a discrete grid world and needs to solve different types of tasks. We customized the lava crossing environment in several ways. We first create a shorter but more narrow crossing path, as the optimal path in deterministic environment. Then we construct a longer but more safe path under perturbation. The map of the mini-grid is shown in Figure 3a. The height of the map can be changed without changing the problem structure and the optimal and robust policy. The state consists of the x-y coordinate of the position, a direction that takes four values, and the time step. The agent only observe a 7×7 area in front of it. (see Chevalier-Boisvert et al. [2018] for details.) We set the observation to be the visual map (RGB image) with a random noise between $(50, 50, 50)$ and $(150, 150, 150)$ for all background pixels. The action space is changed to having five actions: moving forward, turning left, turning right, turning right and moving forward, turning left and moving forward. We set the horizon to three times the number of steps the robust policy needs to reach the goal in the deterministic environment.

Implementation details of `TASID` We describe details of `TASID` and hyper-parameters below. We implement the action predictor model class by a convolutional neural network (CNN). We take the two images x and x' and concatenate them along the channel dimension. This concatenated image is passed through CNN which applies two convolutional layers with ReLu activations followed by a flattening the feature and applying a linear layer to reduce it to a vector of size the number of actions. Finally, we apply a softmax layer to generate probabilities over actions. We train the algorithm with 5×10^5 episodes, for all values of η and gridworld height. We run each experiment 5 times and report averaged results in Figure 3. Shaded bands in the Figure correspond to the 95% Student's t confidence intervals. Other hyperparameters for minigridworld experiments are shown in Table 5.

Amount of compute We run our experiments on a cluster containing P100 GPUs. Each experiment runs on a single GPU in a docker container. We use Python3 and Pytorch 1.4. We found that on P100 GPUs, POTAS took 2 to 6 hours (2 hours for $\eta = 0.1$ and 6 hours for $\eta = 0.5$).

parameter name	values
CNN kernel 1	$8 \times 8 \times 16$, stride 4
CNN kernel 2	$4 \times 4 \times 32$, stride 2
learning rate	0.0003
optimizer	Adam
batch size	256
gradient clipping	100

Table 5: Hyperparameters for `TASID` in the MiniGrid domain