Stealthy Imitation: Reward-guided Environment-free Policy Stealing

Zhixiong Zhuang [1] [2] **Maria-Irina Nicolae** [2] **Mario Fritz** [3]

Abstract

Deep reinforcement learning policies, which are integral to modern control systems, represent valuable intellectual property. The development of these policies demands considerable resources, such as domain expertise, simulation fidelity, and real-world validation. These policies are potentially vulnerable to model stealing attacks, which aim to replicate their functionality using only black-box access. In this paper, we propose Stealthy Imitation, the first attack designed to steal policies without access to the environment or knowledge of the input range. This setup has not been considered by previous model stealing methods. Lacking access to the victim's input states distribution, Stealthy Imitation fits a reward model that allows to approximate it. We show that the victim policy is harder to imitate when the distribution of the attack queries matches that of the victim. We evaluate our approach across diverse, high-dimensional control tasks and consistently outperform prior data-free approaches adapted for policy stealing. Lastly, we propose a countermeasure that significantly diminishes the effectiveness of the attack.[1]

1. Introduction

Neural networks trained with reinforcement learning (RL), known as deep RL policies, are increasingly employed in control systems due to the exceptional performance and automation capabilities. Examples include DeepMind's use of RL in cooling control systems (Luo et al., 2022), throttle valve control in combustion engines (Bischoff et al., 2013),

[1]Graduate School of Computer Science, Saarland University, Saarbrücken, Germany [2]Bosch Center for Artificial Intelligence, Robert Bosch GmbH, Renningen, Germany [3]CISPA Helmholtz Center for Information Security, Saarbrücken, Germany. Correspondence to: Zhixiong Zhuang <zhixiong.zhuang@bosch.com>.

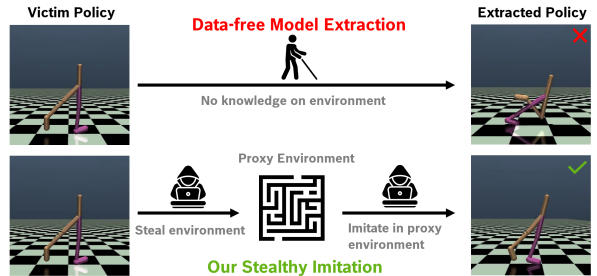[1]The project page is at https://zhixiongzh.github.io/stealthy-imitation.

Figure 1. Traditional data-free model extraction fails in control systems due to the unknown environment with varying sensors. Stealthy Imitation effectively extracts policies by stealing the environment first.

and Festo Robotino XT robot control (Bischoff et al., 2014). Developing a reliable deep RL policy requires substantial resources, including expertise in training, precise simulation, and real-world testing; the resulting policy becomes important intellectual property. However, neural network models are vulnerable to stealing attacks (Tramèr et al., 2016; Orekondy et al., 2019b; Truong et al., 2021) that attempt to copy the functionality of the model via black-box query access. The risks posed by such attacks in control systems are multifaceted, including unauthorized model usage, exposure of sensitive information, and further attacks that can lead to denial of service, operational failures, or even physical damage on the equipment or surroundings.

Model theft typically consists of two steps. First, a transfer dataset is created by querying the victim model with publicly available data (Orekondy et al., 2019b), random noise (Tramèr et al., 2016), or samples synthesized by a neural network (Truong et al., 2021), and recording the model predictions as pseudo-labels. The latter two methods fall under the category of data-free model stealing. After this querying phase, the attackers train their own model via supervised learning, treating the pseudo-labels as ground truth for their samples.

Control systems, such as industrial automation, remotely controlled drones or robots, pose additional challenges for model stealing. A policy perceives states and rewards (also known as the environment), based on which it decides the next action to take. In this context, the attacker can potentially send queries to the system, but does not have access to

the environment. Data-free stealing attacks hold the promise of environment-free policy stealing. While existing data-free attacks have proven effective in the image domain, they operate under the assumption that the attacker knows the valid input range. For instance, valid image pixels are assumed to be in the range of $[0, 255]$. However, such prior knowledge is difficult to acquire in control systems or other applications, due to the distinct semantics and scales of components within the measured state. As a consequence, policy stealing becomes more difficult.

To address this challenge, we introduce Stealthy Imitation (SI), the first environment-free policy stealing attack. Our method solves the two fundamental difficulties of this task: (i) the necessity of accurately estimating the input range and distribution of the states visited by the victim policy, and (ii) the identification of a metric that allows the attacker to evaluate the estimated distribution, and thus its own performance in stealing the policy. These advancements collectively enable a more robust and efficient policy stealing attack. Notably, the derived distribution remains applicable even when the victim updates their policy without altering the training distribution, offering potential savings in query budget for subsequent attacks. Furthermore, it enables the attacker to access confidential information, like sensor types or preprocessing methods. The information could potentially aid in the development of their own control system. The superior performance of our method over traditional data-free model extraction is shown in Figure 1.

Contributions. (i) We introduce a more general and realistic threat model adapted to control systems, where the attacker lacks access to the environment and to the valid input ranges of the policy. (ii) We propose Stealthy Imitation (SI), the first reward-guided environment-free policy stealing method under minimal assumptions. We show our attack to be effective on multiple control tasks. (iii) We introduce the first proxy metric to measure the quality of the estimated distribution. We empirically and statistically validate its correlation with the divergence between the estimated distribution and the actual state distribution of the victim policy. (iv) We develop a defense that is able to counter the proposed attack, thus offering a practical solution for practitioners.

2. Related Work

Knowledge distillation. Knowledge distillation (KD) was initially designed for model compression, aiming to approximate a large neural network (commonly referred to as the teacher model) with a more compact model (the student model). This facilitates deployment on hardware with limited computational capabilities (Ba & Caruana, 2014; Hinton et al., 2015). Unlike our work, which adopts an adversarial view, KD typically presumes access to the teacher model's original training dataset, enabling the stu-

dent model to learn under the same data distribution. When the dataset is large or sensitive, some methods opt for surrogate datasets (Lopes et al., 2017). Others eliminate the need for it by employing data generators in data-free KD approaches (Fang et al., 2019; Micaelli & Storkey, 2019). These methods often assume white-box access to the teacher model for backpropagation, which is a major difference with our setup.

Model stealing. Model stealing focuses on adversarial techniques for the black-box extraction of a victim model (equivalent to the teacher model in KD) (Tramèr et al., 2016; Orekondy et al., 2019b). The attacker, who aims to create a surrogate model (analogous to the student in KD), lacks access to the original training dataset of the victim model. Most existing methods explore data-free stealing, drawing inspiration from data-free knowledge distillation, but lacking the means to use the victim model to train a data generator. These techniques estimate the gradient of the victim model for training their generator and encourage query exploration by synthesizing samples that maximize the disagreement between victim and attacker model (Sanyal et al., 2022; Beetham et al., 2022; Truong et al., 2021). While much work has been conducted in image-based domains, limited research exists on model stealing in the context of reinforcement learning (Behzadan & Hsu, 2019; Chen et al., 2021). Our approach sidesteps the need for environment access and specific knowledge of the RL algorithm employed by the victim. Existing defenses primarily focus on detecting stealing attacks (Juuti et al., 2019; Kesarwani et al., 2018) or perturbing model predictions (Tramèr et al., 2016; Orekondy et al., 2019a). Our proposed defense falls in the latter category: the policy perturbs its outputs when the query falls outside the valid input range.

Imitation learning. Imitation learning aims to train agents to emulate human or expert model behavior. Within this domain, there are two main methodologies. The first is behavioral cloning (BC), which treats policy learning as a supervised learning problem, focusing on state-action pairs derived from expert trajectories (Pomerleau, 1991). The second is inverse reinforcement learning, which seeks to discover a cost function that renders the expert's actions optimal (Russell, 1998; Ng et al., 2000). Another method of interest is generative adversarial imitation learning (GAIL), which utilizes adversarial training to match the imitating agent's policy to that of the expert. Notably, GAIL achieves this alignment using collected data and does not need further access to the environment (Ho & Ermon, 2016). Our work deviates from these imitation learning approaches, as we do not require access to the interaction data between the expert policy, i.e., the victim for us, and its environment.

3. Threat Model

In this section, we formalize the threat model for black-box policy stealing in the context of deep RL policies used in control systems. First, we introduce preliminary concepts and notations. Then, we formalize the victim's policy. Finally, we outline the attacker's knowledge and the relevance of this threat model to real world attacks.

Notations. In the context of deep RL, a policy or agent, is denoted by π with accepting state $\boldsymbol{s}$, and predicting an action $\boldsymbol{a}$, such that $\boldsymbol{a} = \pi(\boldsymbol{s})$. A trajectory $\tau \sim \pi$ consists of a sequence of states and actions collected from the interaction between policy and environment. We represent the initial state distribution as ρ_0, and the environment's state transition function as f, such that $\boldsymbol{s}_{t+1} = f(\boldsymbol{s}_t, \boldsymbol{a}_t)$. The return, or cumulative reward, for a trajectory is represented as $R(\tau)$, while S is the distribution of states visited by the deployed policy.

Victim policy. We consider a victim operating a deep RL policy, π_v, trained to optimize a particular control objective, accepting a state $\boldsymbol{s} \in \mathbb{R}^n$ and predicting the action $\boldsymbol{a}^* \in \mathbb{R}^k$ within the range of $[-1, 1]$ at each time step. Note that victim policies accepting images as input are out of scope, since their input range is typically known (e.g., $[0, 255]$). The environment is fully observable by the victim policy. The performance of the policy is quantified using the expected return $\mathbb{E}_{\tau_v \sim \pi_v}[R(\tau_v)]$ in the deployed setting. S_v represents the distribution of states visited by π_v.

Goal and knowledge of the attacker. We take on the role of the attacker, with the goal of training a surrogate policy π_a, predicting action $\hat{\boldsymbol{a}}$, to replicate the functionality of the victim policy π_v for similar return in the environment. The attacker possesses black-box access to π_v by querying states and obtaining actions as responses. The total amount of queries is represented as B. However, the attacker lacks knowledge on several key aspects: (i) the internal architecture and RL training algorithm of π_v, (ii) the environment setup, including the initial state distribution ρ_0, the state-transition function f, and the reward function R, (iii) the semantics associated with the input and output spaces, (iv) the range of the inputs, as well as the state distribution S_v, and (v) the confidence score of all possible actions from the victim policy. This lack of knowledge makes policy stealing particularly challenging.

Real-world relevance. Our threat model highlights the urgency of addressing vulnerabilities in deep RL policy-based control systems and targets two scenarios where attackers opt for an environment-free policy stealing method to exploit these systems. Firstly, the attacker might not know the environment when they access a networked victim policy they don't own. Secondly, there are cases where interacting with the environment is possible but impractical and ineffi-

cient due to the risk of being detected, time, cost and damage concerns. With the widespread adoption of the Internet of Things (IoT) and control systems' physical world impact, these scenarios are common. Once the policy is stolen, it can lead to additional system attacks, causing service denial, operational failures, or equipment damage. Moreover, successful state distribution estimation can reveal sensitive information like sensor types or preprocessing methods, helping attackers develop their own systems and posing future security threats. Therefore, it is important to investigate how attackers can steal policies without environment access and create defenses to safeguard control systems.

4. Approach: Stealthy Imitation

This section introduces the details of Stealthy Imitation. The method overview in Section 4.1 is followed by an explanation of each of its components in Section 4.2. Section 4.3 shows how to use the estimated distributions from prior steps to steal the target policy. Lastly, we propose a defense that can make the attacker's goal more difficult to reach.

4.1. Method Overview

We introduce Stealthy Imitation as attacker that steals a policy without access to the environment or to the valid input range. To achieve their goal, the attacker aims to optimize the surrogate policy π_a to minimize the expected return difference between their own policy π_a and that of the victim π_v in the environment:

$$\underset{\pi_a}{\arg\min} \left| \underset{\tau_a \sim \pi_a}{\mathbb{E}}[R(\tau_a)] - \underset{\tau_v \sim \pi_v}{\mathbb{E}}[R(\tau_v)] \right| \qquad (1)$$

However, the attacker does not have access to the environment or the reward function. Instead, they can minimize the action difference between their policy and that of the victim on an estimated state distribution S_a using a loss function $\mathcal{L}$ as a proxy for the reward:

$$\underset{\pi_a}{\arg\min} \underset{\boldsymbol{s} \sim S_a}{\mathbb{E}}[\mathcal{L}(\pi_v(\boldsymbol{s}), \pi_a(\boldsymbol{s}))] \qquad (2)$$

The attacker's goal is thus to find both the victim policy and the appropriate distribution of states. The Stealthy Imitation objective encourages exploration by maximizing the disagreement between the victim and attacker models:

$$\overset{\text{II}}{\underset{\pi_a}{\arg\min}} \ \overset{\text{III \& IV}}{\underset{S_a}{\arg\max}} \ \underset{\boldsymbol{s} \sim S_a}{\mathbb{E}} \left[\mathcal{L}(\pi_a(\boldsymbol{s}), \overset{\text{I}}{\pi_v(\boldsymbol{s})}) \right]. \qquad (3)$$

The core of Stealthy Imitation consists of four main steps repeated iteratively until the attacker query budget is consumed: (I) **transfer dataset construction** by querying the
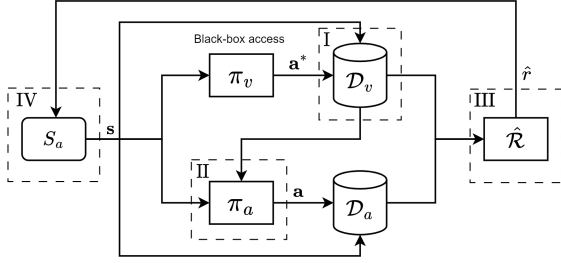
Figure 2. Overview of Stealthy Imitation that iteratively refines the estimated state distribution S_a.

victim policy with states sampled from the estimated distribution S_a; (II) training the attacker policy π_a via **behavioral cloning** to mimic the victim policy on the transfer dataset; (III) **reward model training** $\hat{R}$ to discriminate the behaviours of the victim and current attacker policy, and (IV) **reward-guided distribution refinement** to closer match the victim's state distribution using the proxy reward score on each query state. Once the attacker's budget is exhausted, we train π_a from scratch only on the best estimated distribution with the help of a distribution evaluator. The approach overview is depicted in Figure 2. We detail each step in the following.

4.2. State Distribution Estimation

I. Transfer dataset construction. As the attacker has no knowledge of the state distribution of the victim S_v, we choose a diagonal Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma}^2)$ as estimate of the attacker distribution S_a ($S_a(\boldsymbol{s}; \boldsymbol{\mu}, \boldsymbol{\sigma})$ from here on). States $\boldsymbol{s}$ are sampled from this distribution and passed to the victim policy to obtain corresponding actions $\boldsymbol{a}^*$. The transfer dataset $\mathcal{D}_v$ described below is split into training and validation for use in the subsequent method steps:

$$\begin{aligned} \mathcal{D}_v &= \{(\boldsymbol{s}, \boldsymbol{a}^*)\}, \\ \text{where} \quad \boldsymbol{s} &\sim S_a(\boldsymbol{s}; \boldsymbol{\mu}, \boldsymbol{\sigma}), \\ \text{and} \quad \boldsymbol{a}^* &= \pi_v(\boldsymbol{s}). \end{aligned} \tag{4}$$

Lacking prior knowledge, S_a is initialized with $\boldsymbol{\mu} = \boldsymbol{0}_n$ and $\boldsymbol{\sigma} = \boldsymbol{1}_n$. In each iteration, we use a dynamic query budget by multiplying a base budget b_v with the average of $\boldsymbol{\sigma}$. This ensures sufficient learning in mimicking the actions of the victim policy, especially when the estimated $\boldsymbol{\sigma}$ is large, thereby stabilizing the refinement process.

II. Behavioral cloning. We follow the conventional step in model stealing to mimic the victim policy's behavior using the training split of the transfer dataset $\mathcal{D}_v$. To this end, we employ behavioral cloning using Huber loss (Huber, 1964), which is robust to outliers like L1 and smooth and

differentiable near the minimum like L2:

$$\mathcal{L}_b(\hat{\boldsymbol{a}}, \boldsymbol{a}^*) = \begin{cases} 0.5(\hat{\boldsymbol{a}} - \boldsymbol{a}^*)^2, & \text{if } |\hat{\boldsymbol{a}} - \boldsymbol{a}^*| < 1 \\ |\hat{\boldsymbol{a}} - \boldsymbol{a}^*| - 0.5, & \text{otherwise} \end{cases} \tag{5}$$

III. Reward model training. Our approach is driven by the intuition that the victim policy, while complex within its domain, behaves more simply outside it due to insufficient training. This simpler behavior can be more easily mimicked. Based on this intuition, we hypothesize that as the estimated state distribution S_a approaches the victim's state distribution S_v, the complexity of the victim's responses increases. This makes it more difficult for the attacker policy π_a to accurately imitate the victim. This hypothesis is supported empirically by the results in Section 5.3. To evaluate the difficulty of imitation from the state-action pairs from π_a and π_v, we introduce a reward model $\hat{\mathcal{R}}$, inspired by GAIL (Ho & Ermon, 2016). The role of $\hat{\mathcal{R}}$ is to distinguish between state-action pairs generated by the victim and attacker policy. A more effective distinction suggests that the attacker's policy is more challenging to imitate accurately. To this end, we construct a dataset $\mathcal{D}_a$ using actions $\hat{\boldsymbol{a}}$ generated by $\pi_a(\boldsymbol{s})$ after BC, and train a reward model $\hat{\mathcal{R}}$ by minimizing the loss function:

$$\begin{aligned} \mathcal{L}_r(\boldsymbol{s}, \boldsymbol{a}) = &\mathop{\mathbb{E}}_{(\boldsymbol{s}, \hat{\boldsymbol{a}}) \sim \mathcal{D}_a}[-\log(\hat{\mathcal{R}}(\boldsymbol{s}, \hat{\boldsymbol{a}}))] \\ &+ \mathop{\mathbb{E}}_{(\boldsymbol{s}, \boldsymbol{a}^*) \sim \mathcal{D}_v}[-\log(1 - \hat{\mathcal{R}}(\boldsymbol{s}, \boldsymbol{a}^*))]. \end{aligned} \tag{6}$$

IV. Reward-guided distribution refinement. We use the trained reward model from the previous step $\hat{\mathcal{R}}$ to generate proxy reward values $\hat{r}(\boldsymbol{s}, \boldsymbol{a}^*) = -\log(\hat{\mathcal{R}}(\boldsymbol{s}, \boldsymbol{a}^*))$ for each state-action pair. A high reward value $\hat{r}(\boldsymbol{s}, \boldsymbol{a}^*)$ indicates that the attacker policy fails to effectively mimic the victim, suggesting that the state has higher probability in S_v. These reward values serve as weights for the corresponding samples $\boldsymbol{s}$, which we use to recompute the parameters $\boldsymbol{\mu}'$ and $\boldsymbol{\sigma}'$ of the distribution for the next iteration, as follows:

$$\begin{aligned} \boldsymbol{\mu}' &= \frac{\sum_{(\boldsymbol{s}, \boldsymbol{a}^*) \in \mathcal{D}_v} \hat{r}(\boldsymbol{s}, \boldsymbol{a}^*) \cdot \boldsymbol{s}}{\sum_{(\boldsymbol{s}, \boldsymbol{a}^*) \in \mathcal{D}_v} \hat{r}(\boldsymbol{s}, \boldsymbol{a}^*)}, \\ \boldsymbol{\sigma}'^2 &= \frac{\sum_{(\boldsymbol{s}, \boldsymbol{a}^*) \in \mathcal{D}_v} \hat{r}(\boldsymbol{s}, \boldsymbol{a}^*) \cdot (\boldsymbol{s} - \boldsymbol{\mu}')^2}{\sum_{(\boldsymbol{s}, \boldsymbol{a}^*) \in \mathcal{D}_v} \hat{r}(\boldsymbol{s}, \boldsymbol{a}^*)}. \end{aligned} \tag{7}$$

4.3. Policy Stealing on the Estimated Distribution

Since the attacker has no knowledge of the victim states' distribution S_v, we introduce a model π_e, which we term distribution evaluator. This model helps assess the closeness between the attacker and victim distributions S_a and S_v. π_e is trained via behavioral cloning and is reinitialized in

---

Algorithm 1 Stealthy Imitation

1: **Input:** Victim policy π_v (blackbox access), total budget B, reserved budget B_r, base query budgets b_v and b_a for victim and attacker victims respectively in each iteration
2: **Output:** Trained attacker policy π_a
3: Initialize attacker policy π_a, distribution evaluator π_e, reward model $\hat{\mathcal{R}}$, $\boldsymbol{\mu} \leftarrow \mathbf{0}_n$, $\boldsymbol{\sigma} \leftarrow \mathbf{I}_n$
4: Initialize proxy metric $\tilde{\mathcal{L}} \leftarrow -\infty$, consumed budget $B_c \leftarrow 0$, and to be consumed budget $b_c \leftarrow b_v$
5: **repeat**
6:     // I. Transfer dataset construction
7:     $\mathcal{D}_v \leftarrow \text{QueryAction}(\pi_v, \boldsymbol{\mu}, \boldsymbol{\sigma}, b_c)$
8:     // Evaluate current estimated distribution
9:     $\bar{\mathcal{L}}_b \leftarrow \text{DistributionEvaluate}(\mathcal{D}_v, \pi_e, b_v)$
10:    // Update parameters if current loss exceeds max loss
11:    **if** $\bar{\mathcal{L}}_b > \tilde{\mathcal{L}}$ **then**
12:      $\tilde{\mathcal{D}}, \tilde{\mathcal{L}}, \tilde{\boldsymbol{\mu}}, \tilde{\boldsymbol{\sigma}} \leftarrow \mathcal{D}_v, \bar{\mathcal{L}}_b, \boldsymbol{\mu}, \boldsymbol{\sigma}$
13:    **end if**
14:    // II. Behavioral cloning
15:    $\pi_a \leftarrow \text{BehavioralCloning}(\mathcal{D}_v, \pi_a, b_v \cdot \bar{\boldsymbol{\sigma}})$
16:    // III. Reward model training
17:    $\mathcal{D}_a \leftarrow \text{QueryAction}(\pi_a, \boldsymbol{\mu}, \boldsymbol{\sigma}, b_a)$
18:    $\hat{\mathcal{R}} \leftarrow \text{TrainReward}(\mathcal{D}_a, \mathcal{D}_v, \hat{\mathcal{R}}, b_v \cdot \bar{\boldsymbol{\sigma}})$
19:    // IV. Reward-guided distribution refinement
20:    $\boldsymbol{\mu}, \boldsymbol{\sigma} \leftarrow \text{DistRefine}(\mathcal{D}_v, \hat{\mathcal{R}}, b_v \cdot \bar{\boldsymbol{\sigma}})$
21:    $B_c \leftarrow B_c + b_c$
22:    $b_c \leftarrow \max(b_v, b_v \cdot \bar{\boldsymbol{\sigma}})$
23: **until** $B_c + b_c \geq B - B_r$
24: $\tilde{\mathcal{D}} \leftarrow \tilde{\mathcal{D}} \cup \text{QueryAction}(\pi_v, \tilde{\boldsymbol{\mu}}, \tilde{\boldsymbol{\sigma}}, B - B_c)$
25: $\pi_a \leftarrow \text{BehavioralCloning}(\tilde{\mathcal{D}}, \pi_a, |\tilde{\mathcal{D}}|)$ with reinitialized π_a
26: **return** π_a

---

each iteration to ensure its validation loss $\bar{\mathcal{L}}_b$ measures only the error of the current estimated distribution. Based on our hypothesis, a higher loss $\bar{\mathcal{L}}_b$ is indicative of S_a closely mirroring S_v. We only use b_v samples of the transfer dataset $\mathcal{D}_v$ to train π_e instead of $b_v \times \bar{\boldsymbol{\sigma}}$. This ensures it is only affected by the distribution divergence without the influence of training data size. The total query budget B comprises two parts: the budget used to refine the distribution and the reserved budget B_r for training the final attacker policy on the best-estimated distribution. Once the first part of budget is exhausted, i.e., the algorithm is done iterating over steps I-IV, the parameters $\tilde{\boldsymbol{\mu}}$ and $\tilde{\boldsymbol{\sigma}}$ from the iteration that yielded the highest loss value are used to create an optimized transfer dataset using the reserved query budget B_r. Finally, π_a is subsequently retrained from scratch via BC using this optimized dataset. Algorithm 1 outlines the complete method; all the functions used are defined in Appendix A.

4.4. Stealthy Imitation Countermeasure

Although this work focuses on the attacker's perspective, we also propose an effective defense against Stealthy Imitation. The idea is to leverage the victim's exclusive knowledge of the correct input range; the defender can respond with random actions to invalid queries. We argue that ignoring

queries outside the valid range is not advisable for the victim, as it would leak information about the valid range itself. This approach serves to obfuscate the attacker's efforts to estimate the input range. This defense does not degrade the utility of the victim policy, as it still provides correct answers to valid queries.

5. Experiments

This section presents our empirical results for Stealthy Imitation. We discuss the experimental setup (Section 5.1), followed by a comparison of our proposed method to baselines (Section 5.2) and analyses and ablation studies (Section 5.3). Finally, we show the real-world robot policy stealing in Section 5.4 and the defense performance in Section 5.5.

5.1. Experimental Setup

Victim policies. We demonstrate our method on three continuous control tasks from Mujoco (Todorov et al., 2012): Hopper, Walker2D, and HalfCheetah. The victim policy is trained using soft actor-critic (SAC) (Haarnoja et al., 2018). The victim architecture is a three-layer fully-connected networks (256 hidden units, ReLU activation). The models output a normal distribution from which actions are sampled. These sampled actions are then constrained to the range [-1,1] using tanh. After training, the prediction action given a query state is determined only by the mean of this output distribution. See Appendix B for a complete description of all the tasks and performance of the victim policies. All information about compute resources are summarised in Appendix C.

Attacker policies. Similar to Papernot et al. (2016); Orekondy et al. (2019b;a), we employ the architecture of π_v for π_a, while omitting the prediction of the standard deviation and incorporating tanh activation. Our choice of architecture does not significantly influence the refinement of S_a (see Appendix D), although it does introduce greater variance in the cumulative reward. This phenomenon is attributed to compounding errors, a known issue in imitation learning (Syed & Schapire, 2010; Ross et al., 2011; Xu et al., 2020), where minor training deviations can amplify errors. We set the reserved training budget $B_r = 10^6$ and the base query budget $b_v = 10^5$. Both π_a and π_e share the same architecture and are trained for one epoch per iteration. We use the Adam optimizer (Kingma & Ba, 2015) with a learning rate of $\eta = 10^{-3}$ and batch size of 1024. The final training employs early stopping with a patience of 20 epochs for 2000 total epochs. The reward model $\hat{\mathcal{R}}$ is a two-layer fully-connected network (256 hidden neurons, tanh and sigmoid activations). $\hat{\mathcal{R}}$ is trained with a learning rate of 0.001 for 100 steps. Prior to training, we apply a heuristic pruning process to $\mathcal{D}_v$. Specifically, we remove any state-action pairs $(\boldsymbol{s}, \boldsymbol{a})$ where any component of $\boldsymbol{a}$ equals ± 1,

corresponding to the maximum and minimal action values. This is due to the tanh activation function in victim policy mapping extreme logit values to these limits, which may not reflect typical decision-making but rather the extremes of the function's output range. This pruning step further assists the reward model in correctly identifying the victim policy's state-action pattern.

Baseline attacks. Since our method is the first policy stealing without environment access or prior input range knowledge, we compare it against two approaches: (i) Random: transfer datasets are based on three normal distributions with varying scales, namely $\mathcal{N}(\mathbf{0}_n, \mathbf{1}_n^2)$, $\mathcal{N}(\mathbf{0}_n, \mathbf{10}_n^2)$, and $\mathcal{N}(\mathbf{0}_n, \mathbf{100}_n^2)$; the attacker policy π_a is trained using BC; (ii) data-free model extraction (DFME): we adapt the generator-based DFME (Truong et al., 2021) from image classification to control tasks. Convolutional layers are replaced with fully-connected ones, and tanh activation is replaced by batch normalization with affine transformations.

Evaluation. We consider two performance metrics: Kullback-Leibler (KL) divergence and return ratio. The KL divergence $D_{\mathrm{KL}}(S_v \| S_a)$ measures the discrepancy between the estimated state distribution S_a and the victim's state distribution S_v, an aspect not previously quantified in model stealing. We represent S_v with a reference normal distribution $\mathcal{N}(\boldsymbol{\mu}^*, (\boldsymbol{\sigma}^*)^2)$ estimated from a dataset of 1 million states, $\mathbb{S}_v$, collected from the interaction between the victim policy π_v and the environment. The return ratio, denoted as rr, assesses the stealing performance by dividing the average return generated by the attacker policy in the environment by the average return of the victim policy. The return ratio is the average one derived from eight episodes with random initial state. To account for any variability of SI, we report results with five random seeds in Appendix E.

5.2. Stealthy Imitation Attack Performance

We assess various policy stealing methods, as shown in Figure 3. The measure of $D_{\mathrm{KL}}(S_v \| S_a)$ is specific to our approach (top row), as the Random strategy does not refine a distribution, and DFME focuses on fine-tuning samples. We observe that the gap between S_a and S_v becomes consistently smaller and achieves convergence, even when starting from a high value in HalfCheetah. On average, we achieve an 81% reduction in $D_{\mathrm{KL}}(S_v \| S_a)$ across all environments. Our method substantially outperforms other attacks in terms of return ratio (Figure 3, bottom row). In the Hopper environment, we achieve a return ratio of 97% with just 5 million queries. In contrast, the best competing method, $\mathcal{N}(\mathbf{0}_n, \mathbf{10}_n^2)$, under the same query budget reaches only 70% and quickly falls below 25%. Further details on the performance of the reward discriminator can be found in Appendix F. While the Random $\mathcal{N}(\mathbf{0}_n, \mathbf{1}_n^2)$ baseline shows promise in the Hopper environment with 35 million queries,

it does not maintain this performance as consistently as ours across varying query budgets. DFME fails to steal the victim policy, as it focuses on the near-infinite adversarial samples, restricting exploration. Our method, emphasizing regions instead of individual samples, leads to a more extensive and efficient exploration of the unknown input range. More adaptations of DFME are in Appendix G.

5.3. Analysis

Diagonal Gaussian approximation for complex input distributions. The success of our approach, which uses a diagonal Gaussian distribution to approximate real inputs, is robust to complex input distributions. To support this, we analyze the correlation matrices and the distributions of all variables using real state data from MuJoCo environments. The illustrations in Appendix L disclose significant correlations and non-Gaussian distributions among these variables. These findings substantiate the effectiveness of our method in the presence of coupled input variables and complex distributions. Furthermore, in Section 5.4, we show a high return ratio when applying our method to more realistically modeled robots with higher-dimensional inputs. This provides additional evidence supporting the applicability of our approach across a broad spectrum of input complexities. In Appendix K, we examine the impact of the probabilistic state distribution model and find that a diagonal Gaussian distribution yields better results than a full Gaussian distribution due to the fewer optimized parameters. We also show in Appendix I that RL-trained policies in control systems can be easily compromised using supervised learning when the input distribution is exposed, even through a diagonal Gaussian distribution. Moreover, Appendix H experimentally shows that the diagonal Gaussian approximation is robust to estimation errors on $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$.

Correlation between difficulty of imitation and distribution divergence. To empirically evaluate the hypothesis that the difficulty of imitation is correlated with the divergence between S_a and S_v, we create 600 estimated state distributions S_a. These distributions are parameterized as $S_a(\boldsymbol{s}; \boldsymbol{z}\boldsymbol{\sigma}^* + \boldsymbol{\mu}^*, \boldsymbol{\sigma}^*)$, where each element of $\boldsymbol{z}$ is randomly sampled from a uniform distribution over $[0, 4]$, and its sign is chosen randomly. As a result, the KL divergences $D_{\mathrm{KL}}(S_v \| S_a)$ for these estimated state distributions range approximately from 0 to 8. For each S_a, we construct a transfer dataset of 10^5 points and train the attacker's policy π_a using BC for one epoch. We measure the average validation loss $\bar{\mathcal{L}}_b$ as a proxy for the difficulty of imitation. We apply Spearman's rank correlation test to these measurements, and the results are summarized in Table 1. These results demonstrate a statistically significant correlation for $(\bar{\mathcal{L}}_b, D_{\mathrm{KL}})$, thus supporting the use of π_e as a reliable distribution evaluator in Section 4.3.
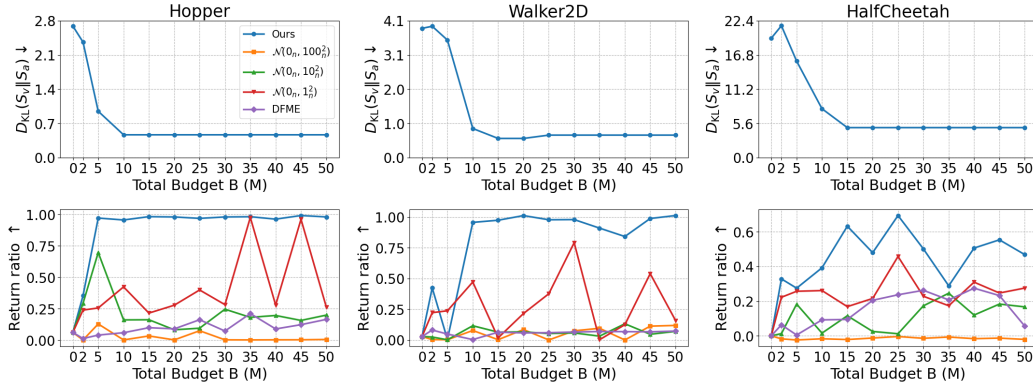
Figure 3. Distribution estimation capacity measured by $D_{\mathrm{KL}}(S_v \| S_a)$ (top) and return ratio (bottom) as a function of the attacker budget.

Table 1. Spearman's rank correlation between validation loss $\bar{\mathcal{L}}_b$ and distribution divergence D_{KL}.

Environment	$(\bar{\mathcal{L}}_b, D_{\mathrm{KL}})$	
	Correlation ρ	p-value
Hopper	-0.84	4.79×10^{-164}
Walker2D	-0.78	7.59×10^{-122}
HalfCheetah	-0.81	4.01×10^{-140}

Ablative analysis. We study the impact of each component of our method by systematically removing them one at a time, while keeping the other components unchanged. The ablation study includes: (i) the use of $b_v \times \bar{\boldsymbol{\sigma}}$ instead of b_v samples of transfer dataset $\mathcal{D}_v$ to train the distribution evaluator π_e; (ii) bypassing the **reward model training** and directly using the validation loss $\mathcal{L}_b$ of each sample as weight for the **reward-guided distribution refinement**; (iii) skipping the pruning step of the transfer dataset before training the reward model; and (iv) using b_v instead of $b_v \times \bar{\boldsymbol{\sigma}}$ to train attacker's policy π_a during **behavioral cloning**. The result is depicted in Figure 4. We observe that incorporating a reward model can more efficiently minimize the distribution divergence $D_{\mathrm{KL}}(S_v \| S_a)$. Additionally, employing a fixed budget for the evaluator model helps the attacker select a better S_a, thereby improving the return ratio. We also note the stabilizing effect of pruning the transfer dataset prior to training the reward model. Moreover, if a dynamic budget is not used when constructing the transfer dataset, we observe undesired shifts in S_a over iterations in Hopper and this leads to a significant reduction in the return ratio.

5.4. Real-world Robot Policy Stealing

We validate Stealthy Imitation in a realistic scenario where the victim policies are trained for the Franka Emika Panda robot simulated by panda-gym (Gallouédec et al., 2021).

The victim policies are from HuggingFace[2] and were developed by independent contributors using truncated quantile critics (TQC) (Kuznetsov et al., 2020). TQC represents a RL algorithm distinct from that utilized in Mujoco. The range of returns observed spanned from -50 to 0.

Experimental setup. Initializing with $\mathcal{N}(\mathbf{0}_n, \mathbf{1}_n^2)$ by chance leaves little room for optimization, as it already results in a very small D_{KL}. To demonstrate efficacy, we initialize the estimated distribution with $\mathcal{N}(\boldsymbol{\mu}^* + 3\boldsymbol{\sigma}^*, (\boldsymbol{\sigma}^*)^2)$, thus with all initial D_{KL} being 4.5. The training involves five epochs for the attacker policy per iteration in the loop, with other hyperparameters mirroring those in the Mujoco setup. We calculate the return ratio using $\frac{R(\tau_a) - R_{min}}{R(\tau_v) - R_{min}}$, where R_{min} is the minimal return -50. We conduct the experiments five times, each with a different random seed. To investigate the influence of query budget to reserve after distribution estimation, namely B_r, we include an additional experiment in which 3 million queries are reserved.

Stealthy Imitation results. Figure 5 illustrates the results, and the task details along with the victim return are summarized in Appendix B. We observed that SI outperforms DFME significantly in two of the Panda robot tasks. When the B_r is 1M, labeled as "SI 1M", SI achieves a high return ratio with only 10M total budget for PandaPickAndPlace and PandaReach, specifically about 70% and 100% respectively. In the more challenging task of PandaSlide, although the return ratio is only approximately 21%, the D_{KL} value significantly decreases by 77%. This lower return ratio could be attributed to the unique characteristic of this task: the robot acts mainly at the start, then waits for the object to hit the target. Despite a good distribution estimation, the attacker policy predominantly clones non-essential actions. We also observe improved performance with increased reserved query budgets for PandaPickAndPlace.
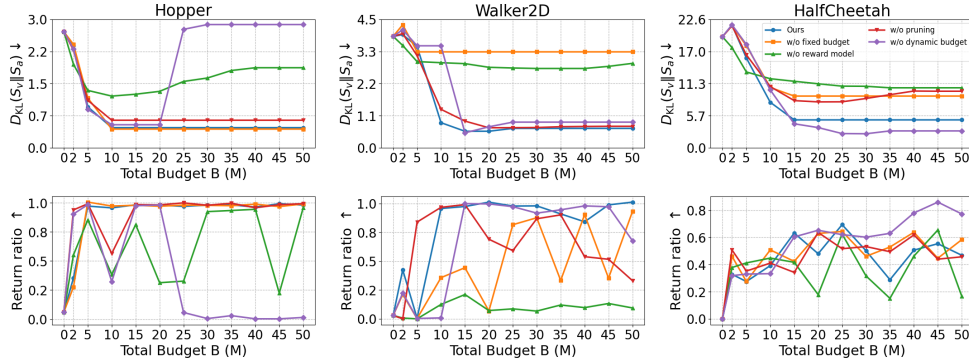
---

[2]https://huggingface.co

Figure 4. We validate the necessity of (i) fixing the dataset size to train the evaluator model, (ii) dynamic budget, (iii) reward model, and (iv) pruning the transfer dataset.
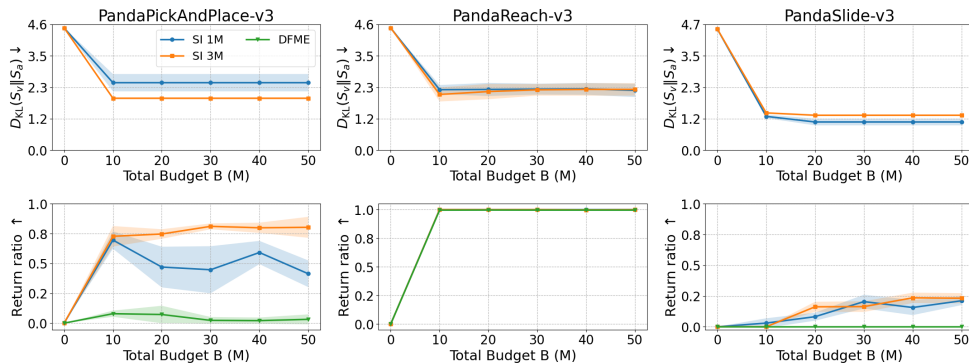

Figure 5. Panda: $D_{\mathrm{KL}}(S_v \| S_a)$ (top) and return ratio (bottom) as a function of the attacker budget.

5.5. Stealthy Imitation Attack Countermeasure

We test the efficiency of the proposed defense to our Stealthy Imitation attack in Mujoco. We consider the input range to match the minimum and maximum values encountered during training. Upon detecting a query that is outside the predefined input range, the victim policy will uniformly sample an action as a response. We present the results in Table 2, evaluating the relative change in KL divergence, denoted as ΔD_{KL}, and the return ratio after exhausting the entire 50M query budget. ΔD_{KL} is calculated as the proportional change from the initial to the final KL value, with negative values indicating that the estimated distribution is converging towards the actual distribution. The results indicate that the countermeasure substantially impedes the attacker's ability to approximate the victim's distribution, consequently reducing the return ratio of the attacker policy. More elaborations are in Appendix J.

6. Discussion

Computational efficiency. In addition to theft effectiveness, Stealthy Imitation also demonstrates computational efficiency. The main computational load comes from train-

Table 2. Results of defense in ΔD_{KL} and return ratio.

Environment	Setup	ΔD_{KL}	Return Ratio
Hopper	w/o defense	−83%	98%
	w/ defense	6%	0%
Walker2D	w/o defense	−83%	101%
	w/ defense	7%	7%
HalfCheetah	w/o defense	−75%	47%
	w/ defense	9%	4%

ing the attacker policy π_a on the optimized transfer dataset $\hat{\mathcal{D}}$. This is more computational efficient compared to utilizing all data with size of total budget B like random strategy.

Limitations and future work. The limitations in this work present opportunities for future research and exploration. Firstly, attackers should consider the potential effects of initial distribution discrepancies. While our method, initializing the estimated distribution with a standard Gaussian, has proven effective, the threshold beyond which initial distribution divergence compromises effectiveness remains to be identified. Secondly, Stealthy Imitation, being agnostic to the victim RL algorithm, can adapt to various victim poli-

cies trained with other RL strategies; however, performance may vary across different RL algorithms and requires further examination. Finally, expanding our approach to other domains, where acquiring the input range is challenging such as those involving feature vectors, holds considerable promise for future research.

7. Conclusion

We show for the first time that an attacker can successfully steal policies in control systems without requiring environment access or prior knowledge of the input range—a strong attack vector that has not been demonstrated or considered in prior research. Lacking access to the victim data distribution, we show that a Gaussian assumption for the attacker query data is sufficient for efficient policy extraction. Our Stealthy Imitation attack outperforms existing methods adapted to policy stealing for a limited-knowledge attacker. We show that it is harder to imitate the victim policy when the distribution of the attack queries increasingly aligns that of the victim, thus allowing an attacker to refine their query distribution. We encourage policy owners to consider the risks of stealing and to use available defenses, such as the one proposed in this paper, to protect their assets.

Acknowledgements

Impact Statement

This paper presents work aimed at raising awareness among developers and those involved in deployment about the risks associated with Machine Learning. While we are demonstrating an attack in this work, we use a hypothetical simulation setup that enables the evaluation of generic attack vectors that do not target particular systems and do not cause harm. Therefore, no responsible disclosure procedure is necessary or even applies to our work. However, given the importance of the problem and the deployment of RL in control systems, we believe it is now the time to evaluate such attacks and inform developers about potential risks. The potential negative societal impacts include the possibility that the proposed attack could be used against real control systems. However, we also propose a defense and encourage policy owners to use it. Similar to commonly published attacks in ML and security and privacy venues, our goal is to provide a novel evaluation that has the goal of, among others, improving the safety of such systems. The authors strictly comply with the ICML Code of Conduct[3].

References

Ba, J. and Caruana, R. Do deep nets really need to be deep? In *Advances in Neural Information Processing Systems (NeurIPS)*, 2014.

Beetham, J., Kardan, N., Mian, A. S., and Shah, M. Dual student networks for data-free model stealing. In *International Conference on Learning Representations (ICLR)*, 2022.

Behzadan, V. and Hsu, W. Adversarial exploitation of policy imitation. *arXiv preprint arXiv:1906.01121*, 2019.

Bischoff, B., Nguyen-Tuong, D., Koller, T., Markert, H., and Knoll, A. Learning throttle valve control using policy search. In *Machine Learning and Knowledge Discovery in Databases: European Conference, (ECML PKDD)*, pp. 49–64. Springer, 2013.

Bischoff, B., Nguyen-Tuong, D., van Hoof, H., McHutchon, A., Rasmussen, C. E., Knoll, A., Peters, J., and Deisenroth, M. P. Policy search for learning robot control using sparse data. In *International Conference on Robotics and Automation (ICRA)*, pp. 3882–3887. IEEE, 2014.

Chen, K., Guo, S., Zhang, T., Xie, X., and Liu, Y. Stealing deep reinforcement learning models for fun and profit. In *ACM ASIA Conference on Computer and Communications Security (ACM ASIACCS)*, pp. 307–319, 2021.

engine Contributors, D. DI-engine: OpenDILab decision intelligence engine. `https://github.com/opendilab/DI-engine`, 2021.

Fang, G., Song, J., Shen, C., Wang, X., Chen, D., and Song, M. Data-free adversarial distillation. *arXiv preprint arXiv:1912.11006*, 2019.

Gallouédec, Q., Cazin, N., Dellandréa, E., and Chen, L. panda-gym: Open-Source Goal-Conditioned Environments for Robotic Learning. *Robot Learning Workshop: Self-Supervised and Lifelong Learning at NeurIPS*, 2021.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning (ICML)*, pp. 1861–1870. PMLR, 2018.

Hinton, G., Vinyals, O., and Dean, J. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

---

[3]https://icml.cc/public/CodeOfConduct

Ho, J. and Ermon, S. Generative adversarial imitation learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2016.

Huber, P. J. Robust estimation of a location parameter. 35 (1):73 – 101, 1964. Institute of Mathematical Statistics.

Juuti, M., Szyller, S., Marchal, S., and Asokan, N. Prada: protecting against dnn model stealing attacks. In *European Symposium on Security and Privacy (EuroS&P)*, pp. 512–527. IEEE, 2019.

Kesarwani, M., Mukhoty, B., Arya, V., and Mehta, S. Model extraction warning in mlaas paradigm. In *Annual Computer Security Applications Conference (ACSAC)*, pp. 371–380, 2018.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015.

Kuznetsov, A., Shvechikov, P., Grishin, A., and Vetrov, D. Controlling overestimation bias with truncated mixture of continuous distributional quantile critics. In *International Conference on Machine Learning*, pp. 5556–5566. PMLR, 2020.

Lopes, R. G., Fenu, S., and Starner, T. Data-free knowledge distillation for deep neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, Long Beach, CA, USA, 2017.

Luo, J., Paduraru, C., Voicu, O., Chervonyi, Y., Munns, S., Li, J., Qian, C., Dutta, P., Davis, J. Q., Wu, N., et al. Controlling commercial cooling systems using reinforcement learning. *arXiv preprint arXiv:2211.07357*, 2022.

Micaelli, P. and Storkey, A. J. Zero-shot knowledge transfer via adversarial belief matching. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

Ng, A. Y., Russell, S., et al. Algorithms for inverse reinforcement learning. In *International Conference on Machine Learning (ICML)*, volume 1, pp. 2, 2000.

Orekondy, T., Schiele, B., and Fritz, M. Prediction poisoning: Towards defenses against dnn model stealing attacks. In *International Conference on Learning Representations (ICLR)*, 2019a.

Orekondy, T., Schiele, B., and Fritz, M. Knockoff nets: Stealing functionality of black-box models. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4954–4963, 2019b.

Papernot, N., Mcdaniel, P., and Goodfellow, I. J. Transferability in machine learning: from phenomena to black-box attacks using adversarial samples. *ArXiv*, abs/1605.07277, 2016.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. Automatic differentiation in pytorch. In *Advances in Neural Information Processing Systems (NeurIPS) Workshop*, 2017.

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

Ross, S., Gordon, G., and Bagnell, D. A reduction of imitation learning and structured prediction to no-regret online learning. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pp. 627–635. JMLR Workshop and Conference Proceedings, 2011.

Russell, S. Learning agents for uncertain environments. In *Annual Conference on Computational Learning Theory*, pp. 101–103, 1998.

Sanyal, S., ini, Addepalli, S., and Babu, R. V. Towards data-free model stealing in a hard label setting. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 15284–15293, 2022.

Syed, U. and Schapire, R. E. A reduction from apprenticeship learning to classification. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2010.

Todorov, E., Erez, T., and Tassa, Y. Mujoco: A physics engine for model-based control. In *International Conference on Intelligent Robots and Systems (IROS)*, pp. 5026–5033. IEEE, 2012.

Towers, M., Terry, J. K., Kwiatkowski, A., Balis, J. U., Cola, G. d., Deleu, T., Goulão, M., Kallinteris, A., KG, A., Krimmel, M., Perez-Vicente, R., Pierré, A., Schulhoff, S., er, Tai, J. J., Shen, A. T. J., and Younis, O. G. Gymnasium. https://github.com/Farama-Foundation/Gymnasium, 2023.

Tramèr, F., Zhang, F., Juels, A., Reiter, M. K., and Ristenpart, T. Stealing machine learning models via prediction {APIs}. In *USENIX Security*, pp. 601–618, 2016.

Truong, J.-B., Maini, P., Walls, R. J., and Papernot, N. Data-free model extraction. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4771–4780, 2021.

Xu, T., Li, Z., and Yu, Y. Error bounds of imitating policies and environments. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 15737–15749, 2020.

A. Algorithms

We now provide a detailed description of each function in Algorithm 1, along with their pseudo code.

Query action. Query action (Algorithm 2) is the function where we obtain the transfer dataset from victim policy and attacker policy. We sample b state vectors from a Gaussian distribution parameterized by $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$, and obtain responses $\boldsymbol{a}$ from the policy π. When $\pi = \pi_v$, the output is dataset $\mathcal{D} = \mathcal{D}_v$; otherwise, it is $\mathcal{D}_a$ when policy is π_a.

Algorithm 2 QueryAction

1: **Input:** Policy π, mean $\boldsymbol{\mu}$ and standard deviation $\boldsymbol{\sigma}$, query budget b
2: **Output:** Dataset D
3: Sample b data points $\boldsymbol{s}$ from $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma}^2)$
4: $\boldsymbol{a} \leftarrow \pi(\boldsymbol{s})$
5: $\mathcal{D} := \{(\boldsymbol{s}_i, \boldsymbol{a}_i) | i = 1, ..., b\}$
6: **return** $\mathcal{D}$

Behavioral cloning. We train policy π to mimic the state-action pair mapping in dataset $\mathcal{D}$ via supervised learning by minimizing the Huber loss, i.e., behavioral cloning in Algorithm 3. Considering that the attacker policy π_a has different dataset size requirement as distribution evaluator π_e using behavioral cloning, we use an additional demand size N to control it.

Algorithm 3 BehavioralCloning

1: **Input:** Dataset $\mathcal{D} = \{(\boldsymbol{s}_i, \boldsymbol{a}_i^*)\}$, policy π, demand size N, epochs E, learning rate η
2: **Output:** Updated policy π
3: Sample N data from $\mathcal{D}$ and split into training and validation $\mathcal{D}_t$ and $\mathcal{D}_v$
4: **for** $e = 1$ to E **do**
5:    **for** each batch $(\boldsymbol{s}, \boldsymbol{a}^*)$ in $\mathcal{D}_t$ **do**
6:       // Compute loss using Huber loss
7:       Calculate loss $\mathcal{L}_b \leftarrow \text{HuberLoss}(\pi(\boldsymbol{s}), \boldsymbol{a}^*)$
8:       // Update model parameters using gradient descent
9:       $\pi \leftarrow \pi - \eta \nabla_\pi \mathcal{L}_b$
10:    **end for**
11: **end for**
12: **return** π

Train reward. We use the code pipeline provided in engine Contributors (2021) to train the reward model in Algorithm 4, except for the additional function PruneData. Reward model is trained for total 400 steps in each iteration with learning rate $\eta = 10^{-3}$.

Prune data. When the action is equal to maximum or minimal value, i.e., extreme action, it is less likely to be the normal action predicted by the victim policy on the real state distribution, as most control systems do not prefer such extreme action. Extreme action value can easily cause instability in control systems. By pruning the transfer dataset shown in Algorithm 5, the reward model can identity the difference of state-action pairs coming from the victim and attacker policies. For instance, if there is a state-action pair whose action is an extreme value, then the reward model tends to identity it as a state-action pair from the attacker, as there is no such data in the transfer dataset querying the victim policy after pruning.

Distribution evaluate. The function described in Algorithm 6 is exactly the same as behavioral cloning, but the final output of the function is the validation loss $\bar{\mathcal{L}}_b$ of evaluator π_e.

Distribution refinement. We apply Equation (7) on the validation split of the transfer dataset to calculate the new $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$, described in Algorithm 7.

Algorithm 4 TrainReward

1: **Input:** dataset $\mathcal{D}_a$ queried from attacker policy, dataset $\mathcal{D}_v$ queried from victim policy, reward model $\hat{\mathcal{R}}$, demand size N, total steps T, learning rate η
2: **Output:** Trained reward model $\hat{\mathcal{R}}$
3: Sample N data from $\mathcal{D}_v$ and split into training and validation $\mathcal{D}_{vt}$ and $\mathcal{D}_{vv}$
4: $\mathcal{D}'_{vt} \leftarrow \text{PruneData}(\mathcal{D}_{vt})$
5: **for** $i = 1$ to T **do**
6:     Sample batch data $(\boldsymbol{s}_v, \boldsymbol{a}_v)$ from $\mathcal{D}'_{vt}$ and $(\boldsymbol{s}_a, \boldsymbol{a}_a)$ from $\mathcal{D}_a$
7:     $L_v \leftarrow -\log(1 - \hat{\mathcal{R}}(\boldsymbol{s}_v, \boldsymbol{a}_v))$
8:     $L_a \leftarrow -\log(\hat{\mathcal{R}}(\boldsymbol{s}_a, \boldsymbol{a}_a))$
9:     // Compute the gradient of the total loss
10:     $\nabla L \leftarrow \nabla(L_v + L_a)$
11:     // Update the reward model
12:     $\hat{\mathcal{R}} \leftarrow \hat{\mathcal{R}} - \eta \nabla L$
13: **end for**
14: **return** The trained reward model $\hat{\mathcal{R}}$

Algorithm 5 PruneData

1: **Input:** Dataset $\mathcal{D}$
2: **Output:** Cleaned Dataset $\mathcal{D}'$
3: $\mathcal{D}' \leftarrow \emptyset$
4: **for** each $(\boldsymbol{s}_i, \boldsymbol{a}_i)$ in $\mathcal{D}$ **do**
5:     **if** no element of $\boldsymbol{a}_i$ equals 1 or -1 **then**
6:         $\mathcal{D}' \leftarrow \mathcal{D}' \cup \{(\boldsymbol{s}_i, \boldsymbol{a}_i)\}$
7:     **end if**
8: **end for**
9: **return** $\mathcal{D}'$

Algorithm 6 DistributionEvaluate

1: **Input:** Dataset $\mathcal{D} = \{(\boldsymbol{s}_i, \boldsymbol{a}_i^*)\}$, policy π, portion size N, epochs E, learning rate η
2: **Output:** validation loss $\bar{\mathcal{L}}_b$
3: Sample N data from $\mathcal{D}$ and split into training and validation $\mathcal{D}_t$ and $\mathcal{D}_v$
4: **for** $e = 1$ to E **do**
5:     **for** each batch $(\boldsymbol{s}, \boldsymbol{a}^*)$ in $\mathcal{D}_t$ **do**
6:         Calculate loss $\mathcal{L}_b \leftarrow \text{HuberLoss}(\pi(\boldsymbol{s}), \boldsymbol{a}^*)$
7:         $\pi \leftarrow \pi - \eta \nabla_\pi \mathcal{L}_b$
8:     **end for**
9: **end for**
10: Calculate average validation loss $\bar{\mathcal{L}}_b$ on $\mathcal{D}_v$
11: **return** $\bar{\mathcal{L}}_b$

B. Environment and Victim Policy

We conducted our experiments on environments sourced from Gymnasium (Towers et al., 2023). The specific environments, along with their version numbers and the performance metrics of the victim policies, are detailed in Table 3. The victim policies are trained using the Ding repository (engine Contributors, 2021), a reputable source for PyTorch-based RL implementations (Paszke et al., 2017). We employ SAC to train the victim policy; hence, the victim policy comprises an actor and a critic model. The actor model receives the state as input and outputs the action distribution, while the critic model receives a concatenated state and action as input and outputs the Q-value. During queries to the victim policy, only the actor model is utilized, outputting the mean of the action distribution as a response. The state observations primarily consist of the positional coordinates and velocities of various body parts. The video of victim policy is in supplementary material.

For the victim policies in the Panda robot stealing setup, we obtain them directly from HuggingFace, instead of training

---

Algorithm 7 DistRefine

1: **Input:** dataset $\mathcal{D}$, reward model $\hat{\mathcal{R}}$, demand size N
2: **Output:** updated $\boldsymbol{\mu}'$ and $\boldsymbol{\sigma}'$
3: Sample N data from $\mathcal{D}$ and split into training and validation $\mathcal{D}_t$ and $\mathcal{D}_v$
4: $\mathcal{D}_v' \leftarrow \text{PruneData}(\mathcal{D}_v)$
5: $\boldsymbol{\mu}' \leftarrow \frac{\sum_{(\boldsymbol{s},\boldsymbol{a})\in\mathcal{D}_v'} \hat{r}(\boldsymbol{s},\boldsymbol{a})\cdot\boldsymbol{s}}{\sum_{(\boldsymbol{s},\boldsymbol{a})\in\mathcal{D}_v'} \hat{r}(\boldsymbol{s},\boldsymbol{a})}$
6: $\boldsymbol{\sigma}'^2 \leftarrow \frac{\sum_{(\boldsymbol{s},\boldsymbol{a})\in\mathcal{D}_v'} \hat{r}(\boldsymbol{s},\boldsymbol{a})\cdot(\boldsymbol{s}-\boldsymbol{\mu}')^2}{\sum_{(\boldsymbol{s},\boldsymbol{a})\in\mathcal{D}_v'} \hat{r}(\boldsymbol{s},\boldsymbol{a})}$.
7: $\boldsymbol{\sigma}' = \sqrt{\boldsymbol{\sigma}'^2}$
8: **return** $\boldsymbol{\mu}'$ and $\boldsymbol{\sigma}'$

Table 3. Mujoco environments and performance of victim policy.

Environment	Observation space	Action space	Victim return
Hopper-v3	11	3	3593±3
Walker2D-v3	17	6	4680±43
HalfCheetah-v3	17	6	12035±61

them ourselves, to simulate a real-world threat. The performance of these victim policies is described in Table 4.

Table 4. Panda environments and performance of victim policy.

Environment	Observation space	Action space	Victim return
PandaPickAndPlace-v3	25	4	-7±4
PandaReach-v3	12	3	-2±1
PandaSlide-v3	24	3	-12±7

C. Compute Resources

In this section, we provide detailed information on the computational resources used for our main experiments. All experiments were conducted on a single NVIDIA GeForce RTX 2080 Ti GPU. The time required to train the victim policies within the Mujoco environment varied depending on the scenario. Specifically, the Hopper and HalfCheetah models were trained in approximately 12 hours, while the Walker2D model required a more extensive duration, taking up to 2 days. Stealthy Imitation completed within 2 hours, irrespective of the query budget. In contrast, the Random strategy's compute time varied from 1 to 12 hours, based on the query budget. This variability arises from the Random strategy's approach of using the entire dataset to train the attacker policy, unlike our method, which only uses the most effective dataset. For DFME, the compute time is linearly related to the query budget, with a completion time of 2 hours at a 50M query budget. Regarding the Panda task, the compute time is approximately 4 hours for each query budget checkpoint.

D. Influence of Model Architecture

We investigate the impact of various attacker policy architectures on performance when executing Stealthy Imitation. Each victim policy utilizes a three-layer fully-connected network with 256 hidden units. To understand the effect of architecture variations, we modify the attacker policies by adjusting the layer numbers to 4, 6, and 10. Furthermore, we conduct experiments with the original layer structure, but reduce the hidden units to 128.

We depict the results on Figure 6. To better understand the impact, except for $D_{\text{KL}}(S_v\|S_a)$ and return ratio, we also provide raw $D_{\text{KL}}(S_v\|S_a)$ on top row, which is the last $D_{\text{KL}}(S_v\|S_a)$ at the end of the iteration, rather than the one selected by distribution evaluator π_e. We observe that the raw D_{KL} of different architecture choices exhibit similar tendencies,

thus the architecture choice has limited impact on the distribution refinement. In the second row of Figure 6, except for Walker2d, the selection of the $D_{\mathrm{KL}}(S_v\|S_a)$ during refining by π_e guarantee an appropriate estimated distribution S_a and low $D_{\mathrm{KL}}(S_v\|S_a)$, preventing the divergence of distribution approximation. However, we observe that the return ratio exhibits higher variance in the third row. This indicates that the return ratio is sensitive when the architecture is different, even when the estimated distribution is closed to the real state distribution. This is also a challenge in the realm of imitation learning, known as compounding errors (Syed & Schapire, 2010; Ross et al., 2011; Xu et al., 2020). Compounding errors imply that even minor training errors can snowball into larger decision errors. In our case, the minor training error comes from different architecture choices.

It is essential to highlight that this issue of compounding errors is predominantly absent in image classification model stealing, where test data points are independently evaluated. Nonetheless, the robustness of the estimation of the underlying distribution S_v in terms of KL divergence underscores the effectiveness of our approach.
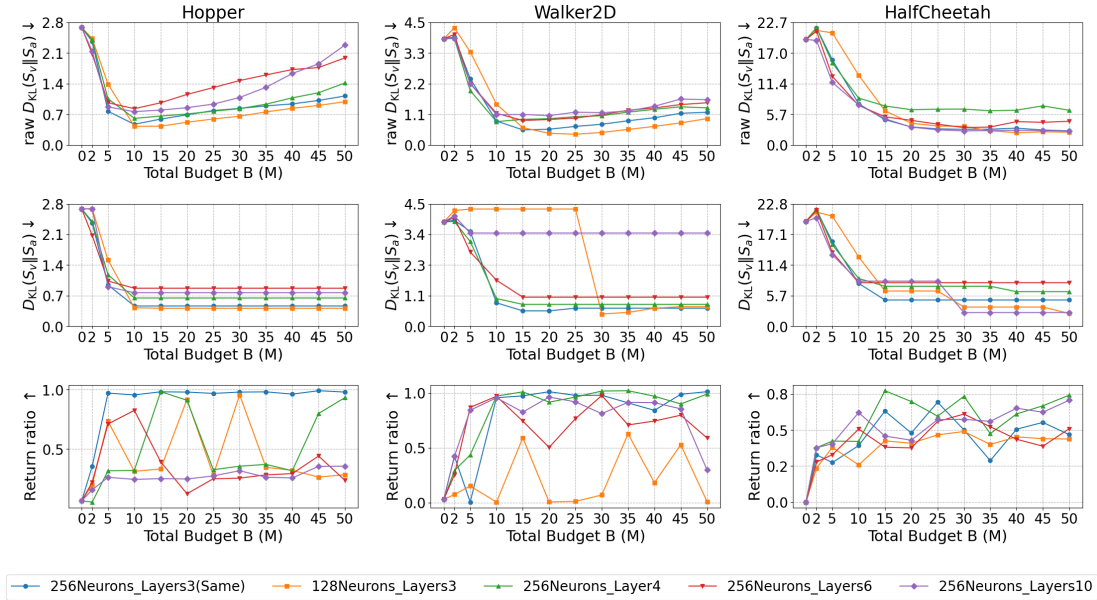

Figure 6. Influence of model architecture on stealing performance.

E. Variability of Stealthy Imitation

We report the variability of Stealthy Imitation in Figure 7 by using five random seeds to obtain five estimated distributions S_a and train five attacker policies π_a. The performance of each policy is still obtained by collecting the average return ratio from eight episodes. We observe that the variability of $D_{\mathrm{KL}}(S_v\|S_a)$ has impact on that of the return ratio, suggesting that a reliable estimated distribution is crucial to attacker policy training. The exact experimental results of Mujoco are listed in Table 5 and Panda in Table 6

F. Performance of the Reward Discriminator

In this section, we analyze how the reward discriminator loss defined in Equation (6) changes throughout the distribution estimation process (Figure 8). In each iteration, we train the reward model for 400 steps; in each step, a batch of data will be sampled from the current victim and attacker distributions, $\mathcal{D}_v$ and $\mathcal{D}_a$ respectively. The x axis in Figure 8 represents the number of steps using a total of 50 million query budget.

We observe that the reward discriminator exhibits oscillations with the variation of the estimated distribution and attacker policy through the iterations. The discriminator's loss may decrease when it successfully identifies attacker's state and action pair data but can increase again as the estimated distribution shifts to a new region where the reward model has not been trained.
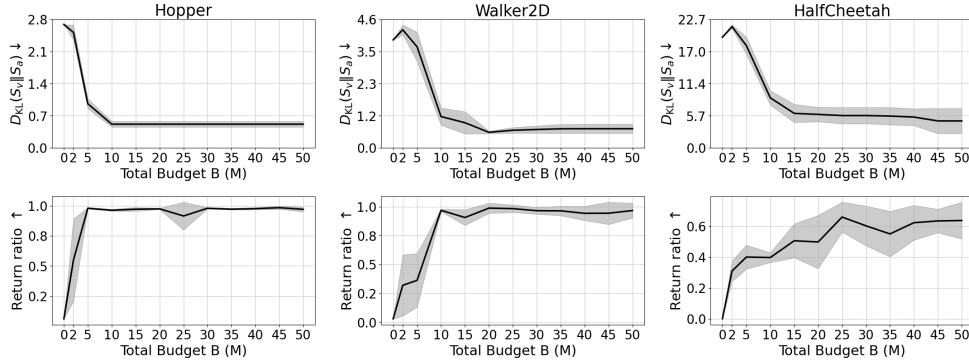
Figure 7. Variability of policy stealing performances.
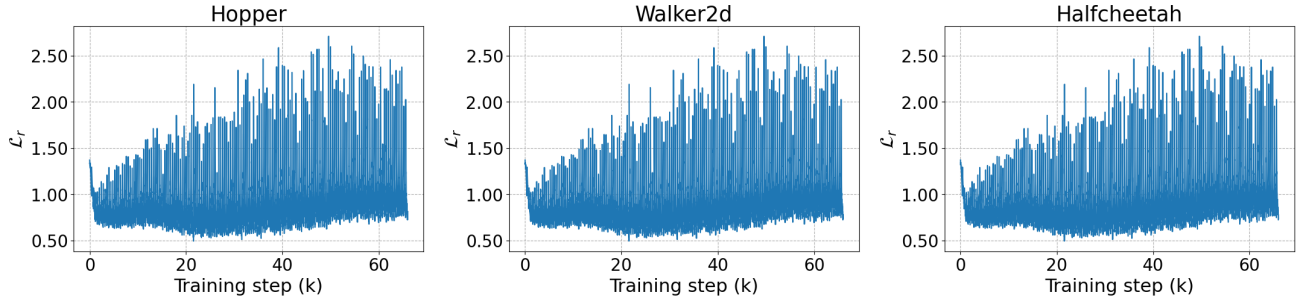

Figure 8. The reward discriminator loss in Equation (6).

G. Adaptation of DFME

Stealthy Imitation is the first method to steal a policy without the knowledge of input range, which means that there are no baselines for this setup. However, we want to provide a comparison to prior art and adapt DFME to the best of our ability to the present setup.

The original DFME generator used a tanh activation function, confining outputs to [-1,1], typical in image classification model stealing. We modified the generator in DFME by either substituting tanh with batch normalization incorporating learnable scaling and shift factors (w/ BN) to enable exploration beyond the initial $\mathcal{N}(\mathbf{0}_n, \mathbf{1}_n^2)$, or by removing tanh without batch normalization (w/o BN). Additionally, we expanded the initial state range by scaling the generator's output by factors of 1, 10, and 100. We conduct the experiments with five different random seeds and report results in Figure 9.
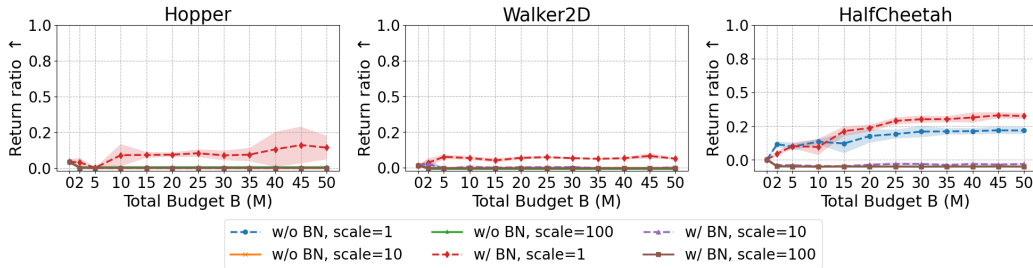

Figure 9. Attacker policy return ratios achieved by adapting DFME with modifications: batch normalization (w/ BN) and without (w/o BN), along with output scaling by factors of 1, 10, and 100. Experiments were conducted using five distinct random seeds.

We find that all attacker policies yield low return ratios, showing DFME's inadequacy in scenarios with unknown input ranges. This stems from DFME's limitation: in any initial state distribution, such as $\mathcal{N}(\mathbf{0}_n, \mathbf{1}_n^2)$, it consistently synthesizes samples, where victim and attacker policies disagree. The infinite amount of adversarial samples restrict its ability to explore

Table 5. $D_{\mathrm{KL}}(S_v \| S_a)$ and return ratio across Mujoco environments with different total budget using Stealthy Imitation.

Metric	Total budget B(M)	Hopper	Walker2D	Halfcheetah
$D_{\mathrm{KL}}(S_v \| S_a)$	0	2.68 ± 0.00	3.87 ± 0.00	19.55 ± 0.00
	2	2.51 ± 0.16	4.23 ± 0.18	21.44 ± 0.32
	5	0.96 ± 0.11	3.62 ± 0.53	18.07 ± 1.52
	10	0.51 ± 0.06	1.12 ± 0.31	8.83 ± 1.27
	15	0.51 ± 0.06	0.90 ± 0.40	6.08 ± 1.61
	20	0.51 ± 0.06	0.55 ± 0.05	5.90 ± 1.27
	25	0.51 ± 0.06	0.63 ± 0.10	5.69 ± 1.44
	30	0.51 ± 0.06	0.65 ± 0.13	5.69 ± 1.44
	35	0.51 ± 0.06	0.68 ± 0.16	5.60 ± 1.54
	40	0.51 ± 0.06	0.68 ± 0.16	5.42 ± 1.49
	45	0.51 ± 0.06	0.68 ± 0.16	4.74 ± 2.22
	50	0.51 ± 0.06	0.68 ± 0.16	4.74 ± 2.22
Return ratio (%)	0	6.15 ± 0.00	2.87 ± 0.00	0.00 ± 0.00
	2	54.89 ± 34.25	31.87 ± 26.44	30.96 ± 6.70
	5	97.90 ± 0.55	36.21 ± 23.27	40.05 ± 7.72
	10	96.25 ± 0.80	96.81 ± 1.10	39.75 ± 3.18
	15	97.07 ± 1.57	90.59 ± 6.79	50.62 ± 11.00
	20	97.39 ± 0.53	98.69 ± 4.42	49.83 ± 17.26
	25	91.47 ± 11.51	98.27 ± 3.23	65.98 ± 9.69
	30	97.87 ± 0.56	96.50 ± 2.95	60.30 ± 12.87
	35	97.13 ± 0.58	96.37 ± 4.09	55.04 ± 14.75
	40	97.57 ± 0.90	94.20 ± 6.15	62.32 ± 11.27
	45	98.33 ± 1.28	94.31 ± 9.61	63.47 ± 7.63
	50	97.00 ± 1.92	96.61 ± 6.35	63.74 ± 11.92

distributions with varying means and scales. In contrast, our Stealthy Imitation method uniquely tackles such problem by analyzing entire regions through a reward model, which evaluates overall regional performance rather than individual samples. We outline the differences between DFME and Stealthy Imitation in Table 7, highlighting how each method addresses distinct threat models and objectives.
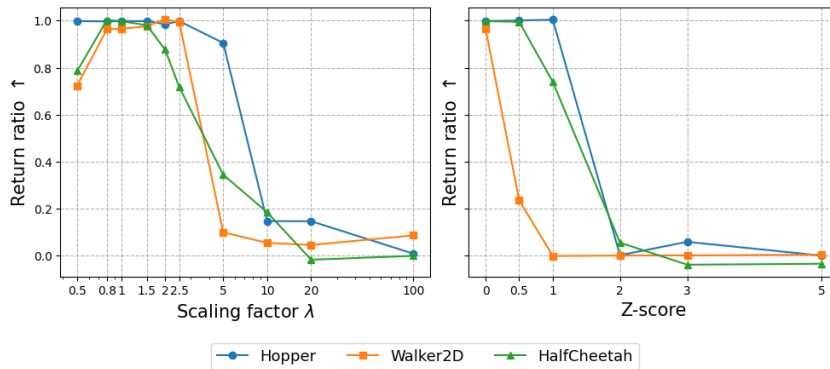
H. Robustness to Distribution Approximation Errors

Figure 10. Left: policy stealing performance (return ratio) when $\boldsymbol{\mu} = \boldsymbol{\mu}^*$ and the scale factor λ modifies $\boldsymbol{\sigma}^*$ such that $S_a = \mathcal{N}(\boldsymbol{\mu}^*, (\lambda\boldsymbol{\sigma}^*)^2)$. Right: policy stealing performance (return ratio) with $\boldsymbol{\sigma} = \boldsymbol{\sigma}^*$ and $\boldsymbol{\mu} = \boldsymbol{z}\boldsymbol{\sigma}^* + \boldsymbol{\mu}^*$, such that $S_a = \mathcal{N}(\boldsymbol{z}\boldsymbol{\sigma}^* + \boldsymbol{\mu}^*, (\boldsymbol{\sigma}^*)^2)$.

Table 6. $D_{\mathrm{KL}}(S_v\|S_a)$ and return ratio across Panda environments with different reserved budget and total budget using Stealthy Imitation.

Metric	Reserved budget(M)	Total budget(M)	PandaPickAndPlace-v3	PandaReach-v3	PandaSlide-v3
$D_{\mathrm{KL}}(S_v\|S_a)$	1	0	4.50 ± 0.00	4.50 ± 0.00	4.50 ± 0.00
		10	2.48 ± 0.32	2.23 ± 0.17	1.26 ± 0.06
		20	2.48 ± 0.32	2.24 ± 0.24	1.05 ± 0.12
		30	2.48 ± 0.32	2.25 ± 0.20	1.05 ± 0.12
		40	2.48 ± 0.32	2.26 ± 0.22	1.05 ± 0.12
		50	2.48 ± 0.32	2.20 ± 0.25	1.05 ± 0.12
	3	0	4.50 ± 0.00	4.50 ± 0.00	4.50 ± 0.00
		10	1.91 ± 0.03	2.05 ± 0.26	1.39 ± 0.00
		20	1.91 ± 0.03	2.16 ± 0.28	1.30 ± 0.00
		30	1.91 ± 0.03	2.22 ± 0.21	1.30 ± 0.00
		40	1.91 ± 0.03	2.24 ± 0.23	1.30 ± 0.00
		50	1.91 ± 0.03	2.24 ± 0.23	1.30 ± 0.00
Return ratio (%)	1	0	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
		10	69.61 ± 7.27	99.99 ± 0.08	2.98 ± 3.91
		20	46.99 ± 17.03	100.00 ± 0.07	8.18 ± 3.26
		30	44.73 ± 19.72	99.99 ± 0.08	20.45 ± 5.57
		40	59.18 ± 9.72	100.03 ± 0.08	15.65 ± 6.15
		50	41.38 ± 11.16	100.05 ± 0.08	21.14 ± 3.55
	3	0	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
		10	72.75 ± 8.57	100.04 ± 0.04	0.00 ± 0.00
		20	74.67 ± 3.99	99.99 ± 0.07	16.22 ± 4.08
		30	81.08 ± 2.46	100.04 ± 0.12	16.38 ± 4.34
		40	79.88 ± 4.29	99.84 ± 0.07	23.52 ± 4.22
		50	80.28 ± 8.77	99.99 ± 0.08	23.22 ± 4.14

Table 7. Contrasting DFME and SI in high level regarding to the query data, target, focus, reusability, and advantage.

Type	DFME	SI
Query data	Model-generated	Probability distribution
Target	Adversarial examples	Hard-to-Imitate regions
Focus	Enhancing sample difficulty using L_1 loss	Assessing overall difficulty via a reward model
Reusability	Limited, dependent on the model	High, as the state distribution is consistent
Advantage	Improves data distribution with input range insight	Effectively determines input range

We customize S_a with different parameters to explore the effect of discrepancy between S_a and S_v. The left of Figure 10 explores the impact of varying $\boldsymbol{\sigma}$ while holding $\boldsymbol{\mu} = \boldsymbol{\mu}^*$ constant such that $S_a = \mathcal{N}(\boldsymbol{\mu}^*, (\lambda\boldsymbol{\sigma}^*)^2)$ with a factor λ. Conversely, the right investigates the effect of modifying $\boldsymbol{\mu}$ while keeping $\boldsymbol{\sigma} = \boldsymbol{\sigma}^*$ constant, $S_a = \mathcal{N}(\boldsymbol{z}\boldsymbol{\sigma}^* + \boldsymbol{\mu}^*, (\boldsymbol{\sigma}^*)^2)$. Different values of $\boldsymbol{z}$ serve as a measure of the divergence between the estimated $\boldsymbol{\mu}$ and $\boldsymbol{\mu}^*$. The sign of each element in $\boldsymbol{z}$ is randomly chosen. Transfer datasets, each containing 1 million queries, are generated from these customized distributions. These datasets are then used to train the attacker's policy π_a through BC for up to 2000 epochs, utilizing early stopping with a patience of 20 epochs. From Figure 10 we observe that minor variations in $\boldsymbol{\sigma}$ are more tolerable compared to deviations in $\boldsymbol{\mu}$.

I. Risk of Exposing Distribution

To demonstrate the risk of exposing the input distribution, we train π_a via behavioral cloning for 200 epochs on five different distributions for S_a, each approximated directly from the real state dataset $\mathbb{S}_v$: (i) and (ii) $\mathcal{N}(\boldsymbol{\mu}^*, (\boldsymbol{\sigma}^*)^2)$ and $\mathcal{N}(\boldsymbol{\mu}^*, \boldsymbol{\Sigma}^*)$: the mean $\boldsymbol{\mu}^*$ and variance $(\boldsymbol{\sigma}^*)^2$ or covariance $\boldsymbol{\Sigma}^*$ are directly calculated from $\mathbb{S}_v$, representing diagonal and full covariance matrix, respectively; (iii) and (iv) $\hat{S}_{v,u}$ and $\hat{S}_{v,m}$: these are non-parametric distribution approximations derived using kernel density estimation (KDE), treating variables as independent and dependent, respectively; and (v) $\mathbb{S}_v$: This samples data
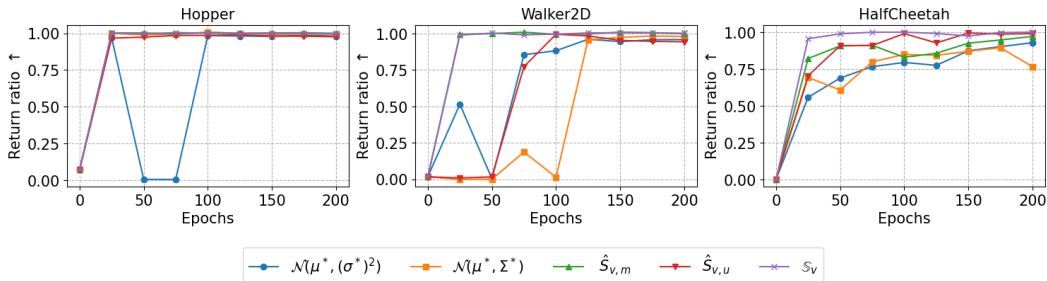
Figure 11. Model stealing success for different choices of S_a based on the underlying distribution S_v.

directly from the real states. Figure 11 shows that successful policy stealing is feasible even when queries are sampled from an approximate distribution, even through a diagonal Gaussian distribution.

J. Discussion on Defense

In this section, we further elaborate on the defense method proposed in Section 4.4. We begin by explaining why the defense works and provide potential strategies for both defenders and attackers, especially when attackers become aware of our defense tactics. We then summarize the key lessons for defenders.

Why does the defense work?   SI is used to estimate the real distribution by observing the difficulty attackers face in mimicking actions across various estimated distributions. A simple yet effective defense against SI involves randomizing outputs for states outside a known input range, thereby increasing the difficulty in distinguishing between different estimated distributions.

How to enhance the defense?   If attackers realize this defense strategy and begin to identify the randomness by analyzing output variance, they might develop new methods of attack. In response, our defense can be enhanced to produce similar variance for both in-range and out-of-range queries. For example, mapping out-of-range queries to a random or fixed point within the range could yield consistent output variance.

What does SI teach us as defenders?   Defenders should be wary of revealing their true input distributions. Such disclosure could potentially expose RL trained policies to the risk of policy stealing via supervised learning.

K. Impact of Probabilistic State Distribution Model

To investigate the impact of the probabilistic state distribution model, we initialize the estimated distribution with not only a diagonal Gaussian distribution but also a full Gaussian distribution, and optimize it in Stealthy Imitation. The experiments are repeated five times with different random seeds. As shown in Figure 12, our method proves effective with different estimated distributions, though it may result in reduced return ratios and increased variance compared with a diagonal Gaussian distribution in Figure 7. This difference primarily stems from the fewer parameters of the diagonal Gaussian, which simplifies the optimization process.
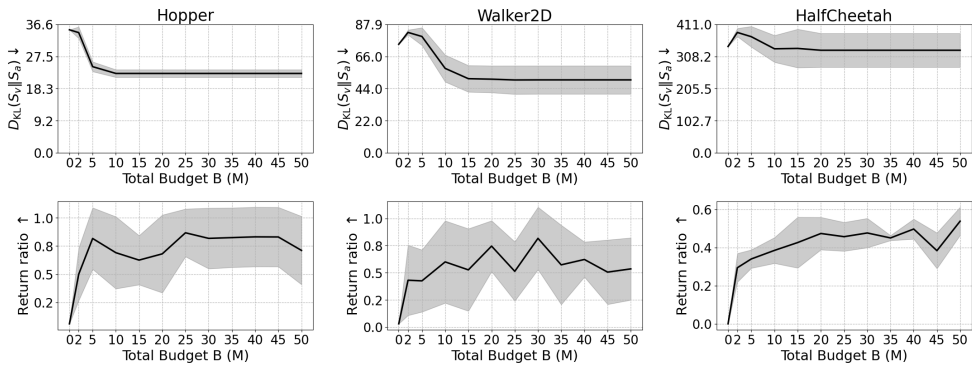
Figure 12. The results of the distribution divergence and the performance of the attacker policy when the estimated distribution is initialized with a full Gaussian distribution and optimized in Stealthy Imitation.

L. Underlying Distribution

In this section, we present a correlation matrix and the distribution shape of the real states, derived from 100k states collected during interactions between the victim policy and the Mujoco environment. We utilize Spearman's rank correlation matrix to analyze the relationships among different variables, as illustrated in Figure 13, Figure 15, and Figure 17. Additionally, we employ KDE, a non-parametric approach, to estimate the probability density functions of these variables, shown in Figure 14, Figure 16, and Figure 18. We observe that the states are highly correlated and cannot be adequately described by a diagonal Gaussian distribution alone. This finding supports the capability of SI to handle more complex input distributions beyond the scope of a diagonal Gaussian.
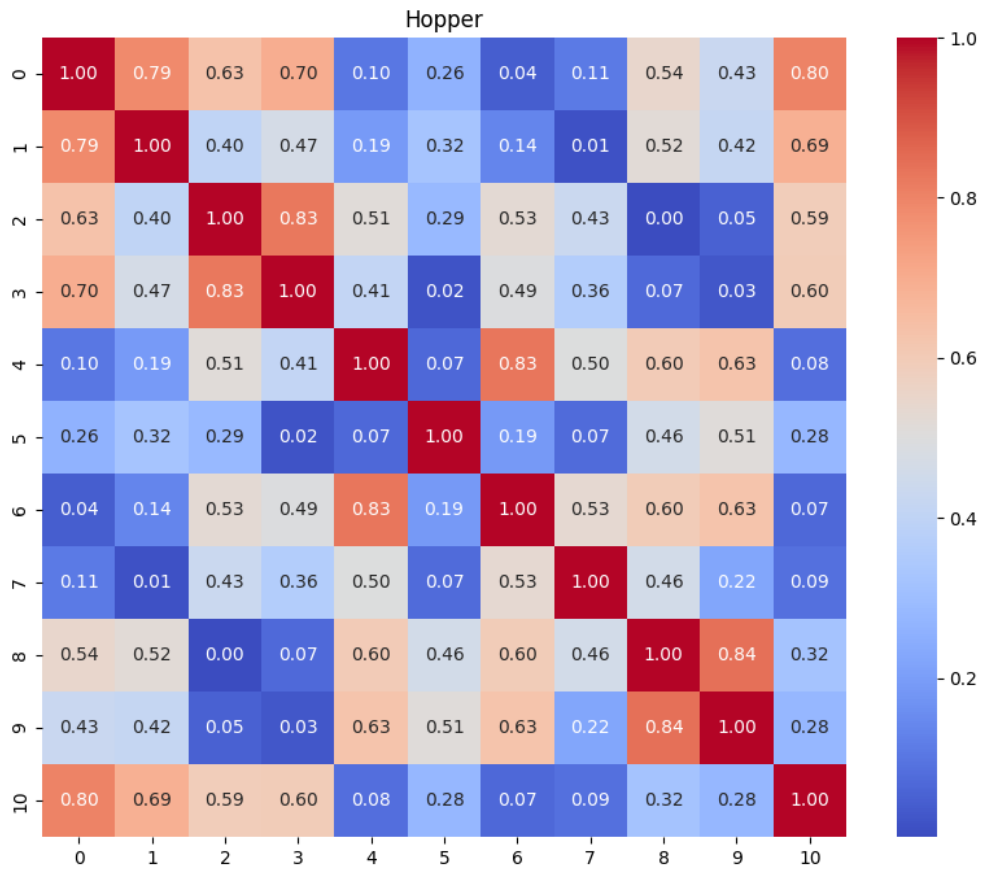
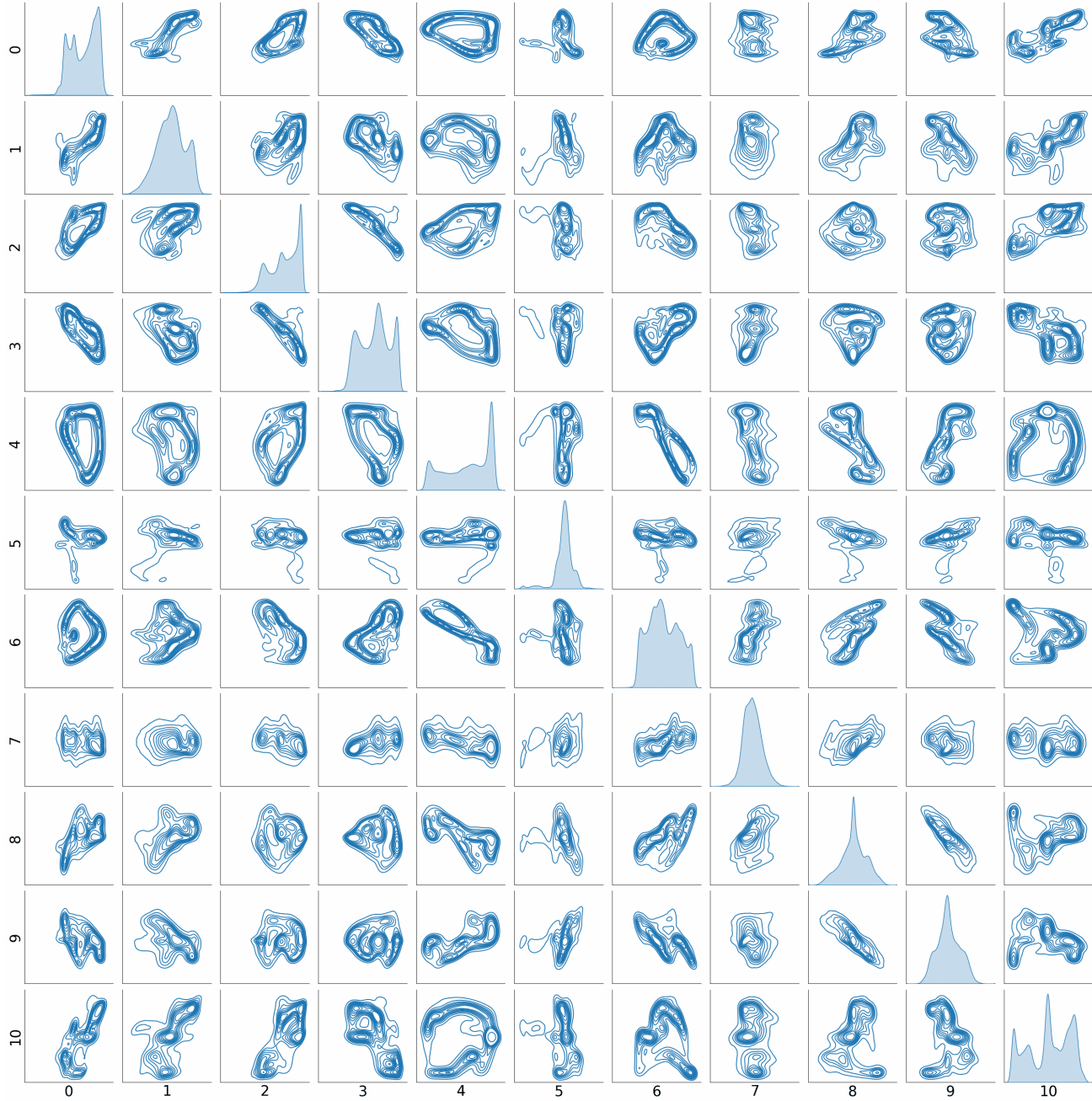*Figure 13.* Correlation matrix of the states collected from Hopper environment.

Figure 14. Distribution visualization using KDE for Hopper environment.
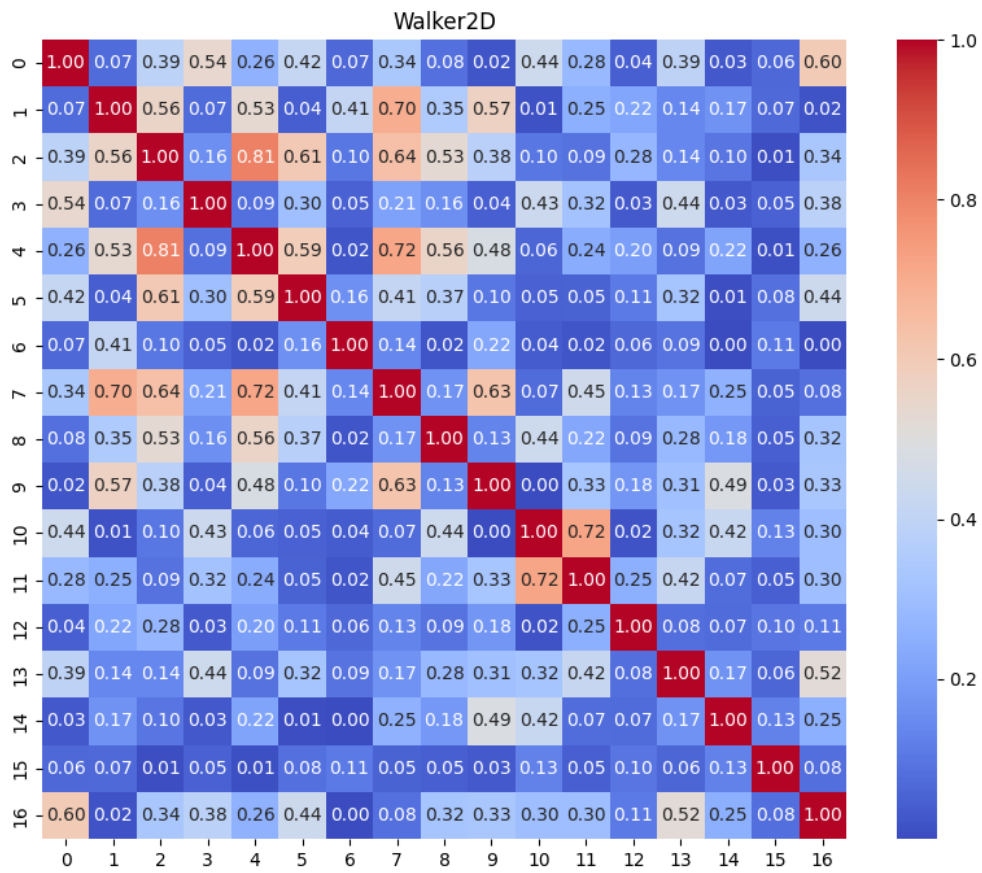
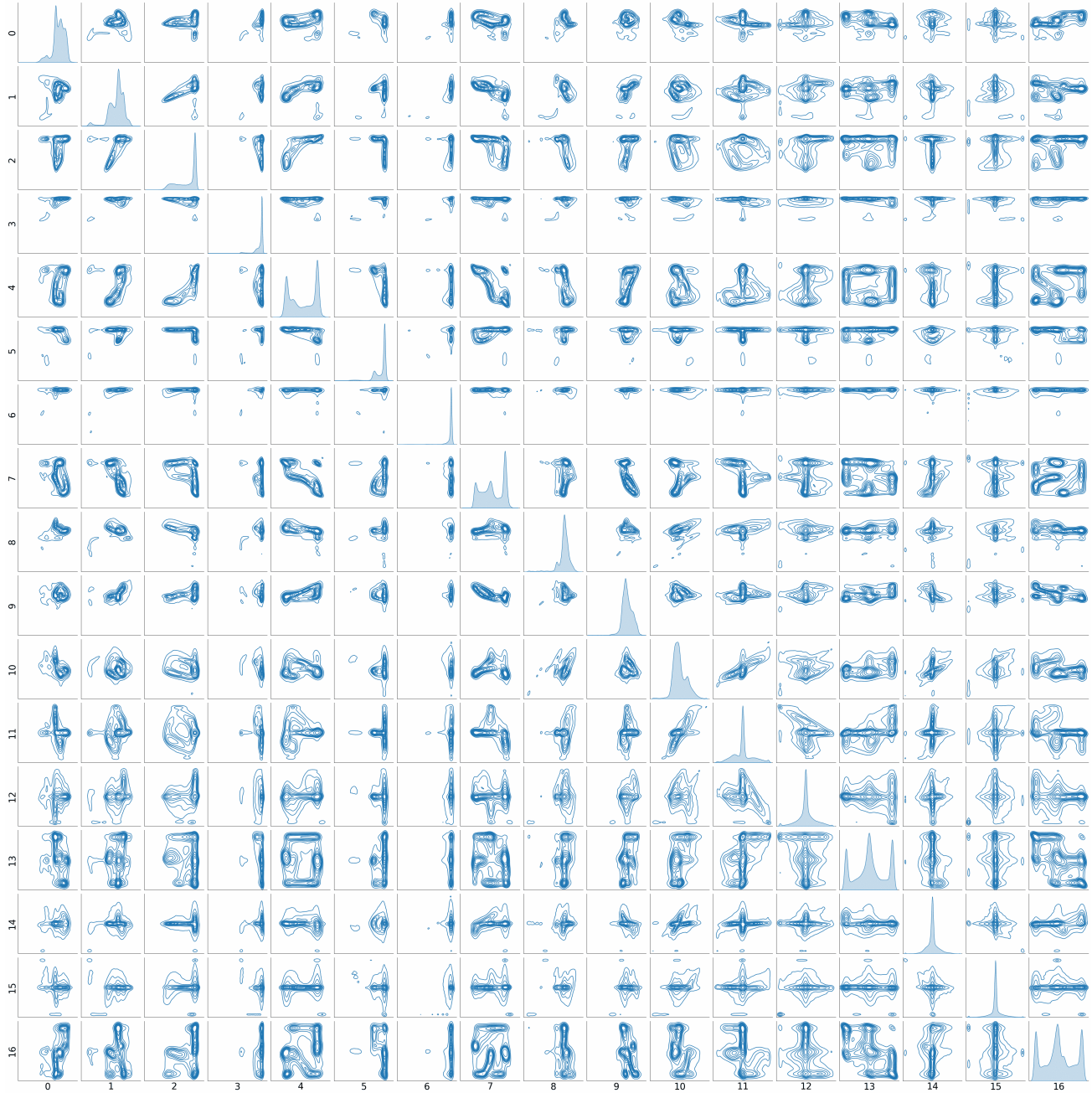*Figure 15.* Correlation matrix of the states collected from Walker2D environment.

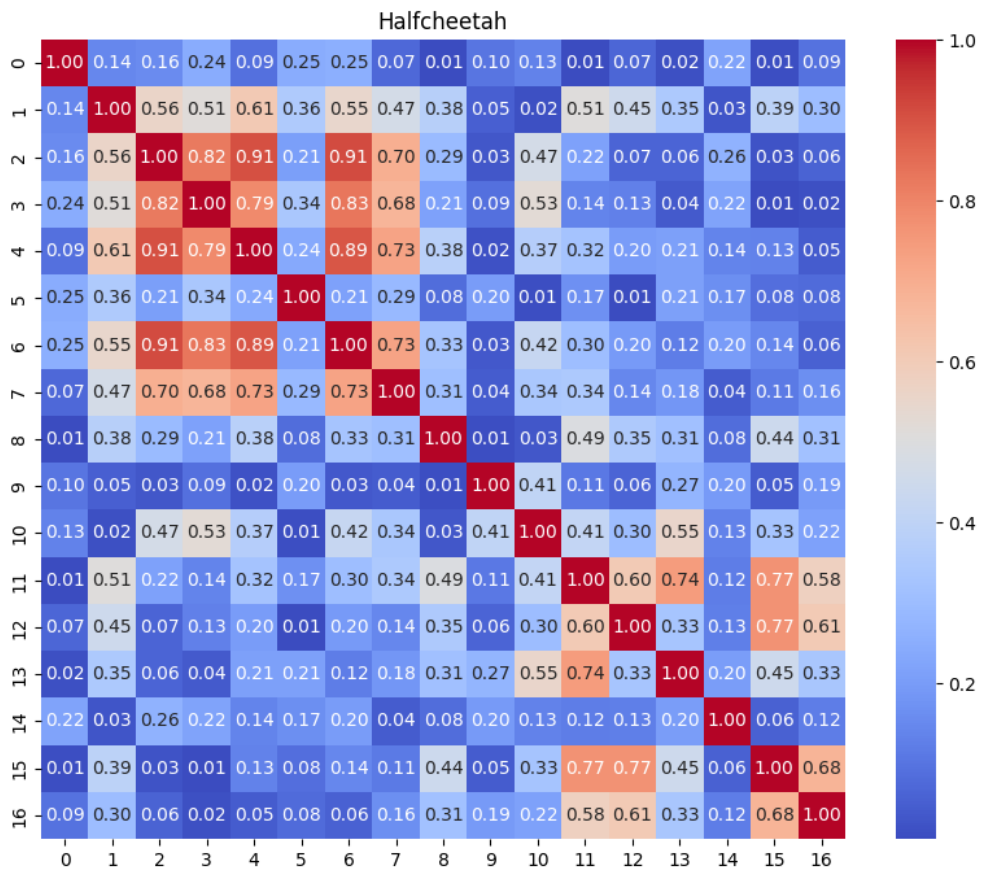*Figure 16.* Distribution visualization using KDE for Walker2D environment.

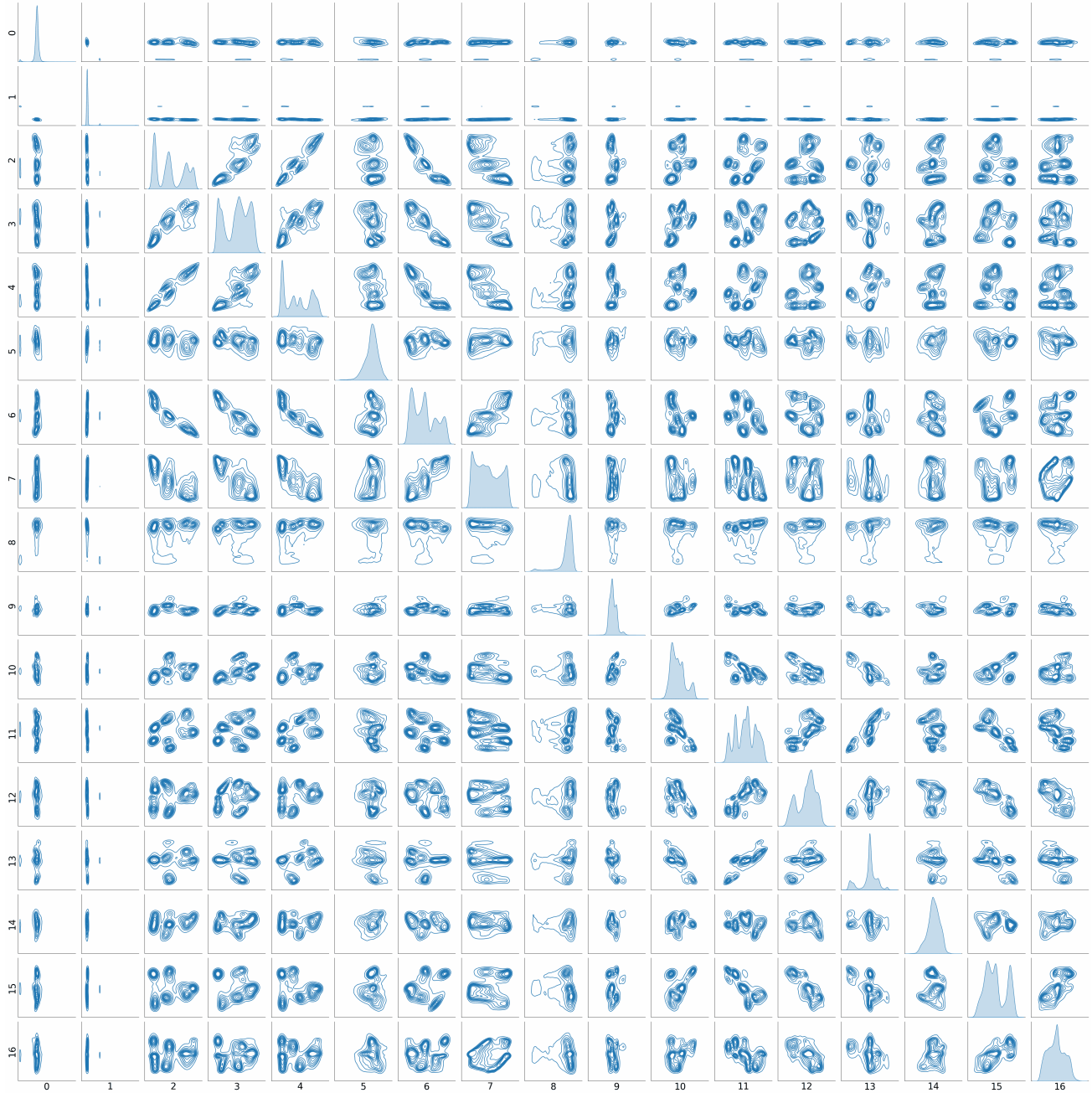*Figure 17.* Correlation matrix of the states collected from HalfCheetah environment.

Figure 18. Distribution visualization using KDE for HalfCheetah environment.