Stealing Deep Reinforcement Learning Models for Fun and Profit

Kangjie Chen
Nanyang Technological University
Singapore
kangjie.chen@ntu.edu.sg

Shangwei Guo*
Chongqing University
China
swguo@cqu.edu.cn

Tianwei Zhang
Nanyang Technological University
Singapore
tianwei.zhang@ntu.edu.sg

Xiaofei Xie
Nanyang Technological University
Singapore
xfxie@ntu.edu.sg

Yang Liu
Nanyang Technological University
Singapore
yangliu@ntu.edu.sg

ABSTRACT

This paper presents the *first* model extraction attack against Deep Reinforcement Learning (DRL), which enables an adversary to precisely recover a black-box DRL model only from its interaction with the environment. Model extraction attacks against supervised Deep Learning models have been widely studied. However, those techniques cannot be applied to the reinforcement learning scenario due to DRL models' high complexity, stochasticity and limited observable information. We propose a novel methodology to overcome the above challenges. The key insight of our approach is that the process of DRL model extraction is equivalent to *imitation learning*, a well-established solution to learn sequential decision-making policies. Based on this observation, our method first builds a classifier to reveal the training algorithm family of the targeted DRL model only from its predicted actions, and then leverages state-of-the-art imitation learning techniques to replicate the model from the identified algorithm family. Experimental results indicate that our methodology can effectively recover the DRL models with high fidelity and accuracy. We also demonstrate two use cases to show that our model extraction attack can (1) significantly improve the success rate of adversarial attacks, and (2) steal DRL models stealthily even they are protected by DNN watermarks. These pose a severe threat to the intellectual property protection of DRL applications.

CCS CONCEPTS

• **Security and privacy**; • **Computing methodologies** → **Machine learning**; **Reinforcement learning**;

KEYWORDS

Model extraction, Deep reinforcement learning, Imitation learning

ACM Reference Format:
Kangjie Chen, Shangwei Guo, Tianwei Zhang, Xiaofei Xie, and Yang Liu. 2021. Stealing Deep Reinforcement Learning Models for Fun and Profit. In *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security (ASIA CCS '21), June 7–11, 2021, Virtual Event, Hong Kong*. ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3433210.3453090

1 INTRODUCTION

Deep Reinforcement Learning (DRL) has gained popularity due to its strong capability of handling complex tasks and environments. It integrates Deep Learning (DL) architectures and reinforcement learning algorithms to build sophisticated policies, which can accurately understand the environmental context (*states*) and make the optimal decisions (*actions*). Various algorithms and methodologies have been designed to facilitate the applications of DRL in different artificial intelligent tasks, e.g., autonomous driving [35], robot motion planning [66], video game playing[40, 49, 65], etc.

As DRL has been widely commercialized (e.g., autonomous driving Wayve [2], path planning MobilEye [1]), it is important for model owners to protect the intellectual property (IP) of their DRL-based products. DRL models are generally deployed as black boxes inside the applications, so the model designs and parameters are not disclosed to the public, which prevents direct plagiarization or modification by malicious users. From the adversarial perspective, we want to investigate the following question in this paper: *is it possible to extract the proprietary DRL model with only oracle access?* This is known as model extraction attack, which has been widely studied for supervised DL models [14, 42, 53]. However, the possibility and feasibility of extracting DRL models have not been explored yet. We make the first step towards this goal.

Although various extraction techniques were designed against supervised DL models, challenges arise when applying them to DRL models due to significant differences of model features and scenarios. First, some attack approaches can only extract very simple models (e.g., two-layer neural networks [31]) and datasets (e.g., MNIST [42]). In contrast, DRL models usually have more complicated and deeper network structures to handle complex tasks, which cannot be extracted by the above techniques. Second, the adversary in the DRL environments has less observable information for model extraction. Past works assume the adversary has access to the prediction confidence scores [31, 42, 53], gradients [38] or the side-channel execution characteristics [4, 26]. In our black-box DRL

[*] Corresponding author

ASIA CCS '21, June 7–11, 2021, Virtual Event, Hong Kong
© 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-8287-8/21/06…$15.00
https://doi.org/10.1145/3433210.3453090

scenario, the adversary can only observe the predicted actions from the DRL model, which are not enough to recover the DRL model with the above methods. Third, supervised DL models perform predictions over discrete input samples, which are independent of each other. However, DRL is designed to solve Markov Decision Process (MDP) problems. Individual input samples cannot fully reflect the inherent features and behaviors of DRL models. The adversary will lose the information of temporal relationships if he only observes these discrete data. Besides, compared to supervised DL models, DRL models are more stochastic and their behaviors highly depend in the environments with different transition probabilities.

In this paper, we propose a novel model extraction methodology for DRL models, which can overcome the above challenges. The key insight of our methodology is that *the process of DRL model extraction is equivalent to an imitation learning task*. Imitation learning [30] is a promising technique to learn and mimic the behaviors of an expert instead of training a policy directly based on a reward function. This exactly matches the scenario of DRL model extraction, which is to mimic the behaviors of the targeted model. We formalize the DRL model extraction attack, and prove its equivalence with imitation learning.

Although there exists such a close connection, it is not easy to directly apply imitation learning for DRL model extraction. Given a task, DRL policies obtained from different training algorithms have significant differences in terms of behaviors and performance (as discussed in Section 2.2). In imitation learning, a DRL algorithm needs to be specified as an oracle. If the adversary does not know the training algorithm employed by the targeted model (which is assumed in our threat model), he will have to randomly pick one algorithm as the basis of imitation learning. The imitated model will behave quite differently if the adversary selects a different algorithm from the victim model.

Our solution addresses this issue by identifying the algorithm family of the DRL model first. We build an algorithm classifier, which is able to predict the algorithm family of a black-box DRL model based on its behaviors and environmental states. Specifically, (1) we use timing sequences of actions as the feature of a DRL model to characterize its decision process and interaction with the environment. (2) We utilize Recurrent Neural Networks as the structure of the classifier for training and prediction, which can better understand the temporal relationships inside the feature sequence. (3) For each DRL model, we generate different feature sequences in environments initialized with different random seeds. This guarantees that the training set of the classifier is comprehensive and includes different behaviors of the same algorithm family.

With the identified training algorithm, we are now able to use state-of-the-art techniques from the imitation learning community to conduct DRL model extraction attacks. Specifically, we adopt Generative Adversarial Imitation Learning (GAIL) [23], which trains a discriminative model and generative model to imitate the behaviors of the targeted DRL policy. The contest between these two models can guarantee that our replication has very similar behaviors as the targeted one on the same environment.

Our attack solution can extract models with high similarity of training algorithm families, behaviors and performance as the targeted models. Extensive experiments show the adversary can achieve 100% of performance accuracy, and 97% of behavior fidelity for various tasks and algorithms. We further provide two use cases to demonstrate that this attack can (1) significantly enhance the adversarial attacks by increasing the transferability of adversarial examples, and (2) easily invalidate the IP protection of watermarking mechanisms. This can bring severe economic loss as well as safety threats to the DRL-based applications. We expect this study can raise people's awareness about the privacy threats of DRL models, as well as the necessity of defense solutions.

The key contributions of this paper are:

- We formally define DRL model extraction and prove the equivalence between this attack and imitation learning.
- We propose a novel method to identify the algorithm family of a black-box DRL model.
- We propose an end-to-end DRL model extraction attack based on imitation learning.
- We conduct extensive experiments and case studies to illustrate the effectiveness and severity of our attack method.

The rest of this paper is organized as follows. Section 2 introduces the background of DRL. Section 3 presents the threat model and formal definition of DRL model extraction. We also give formal analysis of its equivalence with imitation learning. Section 4 exhibits our DRL extraction attack. Section 5 evaluates the effectiveness of the proposed attack, followed by two case studies in Section 6. Section 7 discusses some open issues. We summarize related works in Section 8 and conclude in Section 9.

2 BACKGROUND

2.1 Reinforcement Learning

Reinforcement Learning (RL) is a type of machine learning technology that enables an agent to interact with an environment and learn an optimal policy by maximizing the cumulative reward from the environment. A RL problem can be modeled as a Markov Decision Process (MDP), represented as a tuple $(\mathbb{S}, \mathbb{A}, \mathbb{P}, r, \gamma)$, where

- $\mathbb{S}$ is a finite state space, which contains all the valid states in the environment;
- $\mathbb{A}$ is a finite action space, from which the agent chooses an action as the response to the state it observes;
- $\mathbb{P} : \mathbb{S} \times \mathbb{A} \times \mathbb{S} \rightarrow [0, 1]$ is the state transition probability. For two states s, s' and an action a, the output of $\mathbb{P}$ denotes the probability that s is transited to s' by taking action a;
- $r(s, a)$ is the reward function that outputs the expected reward if the agent takes action a at state s;
- $\gamma \in [0, 1)$ is the discount factor that denotes how much the agent cares about rewards in the distant future relative to those in the immediate future. A smaller factor values places more emphasis on the immediate rewards.

A RL policy $\pi : \mathbb{S} \times \mathbb{A} \rightarrow [0, 1]$ describes the behaviors of an agent in an MDP. It denotes the probability of an action $a \in \mathbb{A}$ the agent will take on a state $s \in \mathbb{S}$. Then the goal of a reinforcement learning is to identify a policy π^* that maximizes the expected cumulative rewards:

$$\pi^* = \arg\max_{\pi} \sum_{t=t_0}^{T} \sum_{a_t \in \mathbb{A}(s_t)} \gamma^{t-t_0} r(s_t, a_t) \pi(s_t, a_t) \qquad (1)$$

where T is the termination timestep. In practice, instead of observing π's action distribution probability on a certain state s, we usually only capture π's optimal action a, and thus the policy can be formulated as $\pi(s) = a$.

2.2 RL with Deep Neural Networks

Despite RL has been studied for a long time and achieved tremendous success in some tasks [41], traditional approaches to solve the RL problem lack scalability and are inherently limited to relatively simple environments. Deep Reinforcement Learning is then introduced, which adopts Deep Neural Networks (DNNs) to understand and interpret complex environmental states, and make the optimal decisions. Due to the great capabilities of neural networks in learning high-dimensional feature representations and function approximation properties, DRL can achieve outstanding performance in mastering human-level control policies in various tasks with high-dimensional states [35, 66]. There are generally three common approaches to solve reinforcement learning tasks.

Value-based Approach. The agent performs certain actions according to its policy to maximize its reward. The optimal behaviors of the policy π are defined by the Q-function which obeys the following Bellman equation,

$$Q^{\pi}(s,a) = \mathbb{E}[r(s,a) + \gamma \max_{a'} Q^{\pi}(s',a')]. \tag{2}$$

This equation shows the maximum return value $Q^{\pi}(s,a)$ from state s and action a is the sum of the immediate reward r and the return obtained following the optimal policy until the end of the episode. When the agent interacts with the environment and transits from state s to the next one s', this approach estimates the value of $Q^{\pi}(s,a)$. Once we obtain all the values of each state-action pair, we can select the optimal action a^* with the highest Q value on the current state s (i.e., $a^* = \arg\max_a Q^{\pi}(s,a)$).

For most problems, however, it is impractical to represent the Q-function as a table containing the values of all possible combinations of states and actions. Deep Q-Network (DQN) was introduced to approximate the Q-value for each action. This algorithm has been extensively used to play GO [50] and Atari games (at superhuman level) [40]. However, DQN cannot be adopted in the tasks with continuous action space since the algorithm requires to learn all the possible Q values.

Policy-based Approach. This solution attempts to identify the optimal policy directly other than estimating all the state-action values. Typical examples include REINFORCE [57] which regards an RL policy as a function $\pi_\theta(s,a) = \mathbb{P}(a|s,\theta)$ and optimizes it by applying the policy gradient technique. To extend this approach to complex tasks, researchers modeled the policy π_θ with DNNs such as Multilayers Perceptron and Convolutional Neural Networks. The objective function of the policy network is defined as the expectation of the total discounted rewards on all the states of a trajectory in an episode,

$$J(\theta) = \mathbb{E}_{(s_t,a_t)\sim\pi_\theta}[\sum_{t=0}^{\infty} \gamma^t r_t], \tag{3}$$

This approach has some limitations. Since the expected reward depends on all the states within the episode, if the agent receives a high reward, it tends to conclude all the actions taken on all the states were good, even if some of them were really bad. Moreover, as the network can only be updated after one episode is completed and the sample of one trajectory can be used for only once, data collection and sample utilization are inefficient in this approach.

Actor-Critic Approach. This is an effective method to overcome the common drawbacks of policy-based methods. It learns both a policy (actor) and a state value function (critic) to reduce variance and accelerate learning. An actor is formed with a policy network, similar as the policy-based approach, which performs action a on the current state s. The critic is a Q-function represented by a network to estimate how good the action a given by the actor at state s is. As specified in [39], the model can be learned with two objective functions: a) the objective function of the actor is the same as Equation 3; b) the advantage function $A\pi_\theta(s,a)$ of the critic represents the extra reward the agent gets if it takes this action:

$$A\pi_\theta(s,a) = Q_{\pi_\theta}(s,a) - (r + \gamma \max_{a'} Q_{\pi_\theta}(s',a')). \tag{4}$$

Therefore, the actor-critic method combines the advantages of both policy-based and value-based methods. The actor enjoys the benefits of computing continuous actions without the need for optimization on a Q-function. The critic's merit is that it supplies the actor with low-variance knowledge of the performance. These properties make the actor-critic methods an attractive reinforcement learning solution. State-of-the-art algorithms include Proximal Policy Optimisation (PPO) [48], Actor-Critic with Experience Replay (ACER) [56] and Actor Critic using Kronecker-Factored Trust Region (ACKTR) [58].

3 ATTACK FORMULATION AND ANALYSIS

3.1 Threat Model

We consider a standard scenario of model extraction, where the adversary has oracle access to the targeted model. Specifically, the adversary can operate the DRL model in the normal environment without manipulating the states. He can observe the states and the model's corresponding actions. This is common when the DRL application is deployed in the open and public environment. For instance, in DRL-based robotics, the adversary can deploy the robot in certain environments, and observe its corresponding reactions. In the application of video game playing, the adversary can start a DRL player with different situations, and collect its strategies.

Different from prior works of extracting supervised DL models [7, 31, 42, 53], we consider more restricted capabilities for the adversary: he has *no* knowledge about the targeted DRL model, including the model structures and parameter values, training algorithms and hyperparameters, etc. He is *not* able to obtain the confidence score of each possible action predicted by the targeted DRL policy. This makes the attack more practical, yet more challenging as well.

3.2 Formal Definition of DRL Model Extraction

According to [31], there are two basic categories of model extraction attacks. (1) *Accuracy extraction* aims to reproduce a model which can match or exceed the accuracy of the targeted model. The replicated model does not need to match the predictions of the oracle precisely. (2) *Fidelity extraction* attempts to recover a model with the same behaviors as the victim one, even for incorrect prediction results.

In this paper, we focus on both accuracy and fidelity extraction of DRL models, which is more threatening.

It is worth noting that we only consider to extract an approximate copy of the targeted model with similar rewards and behaviors, rather than the exact values of model parameters. Parameter extraction is difficult for complex DL models [31], as different values can give the same model outputs, which are indistinguishable from the adversary. Besides, the adversary cannot identify the values of dead neurons which never contribute to the model output.

Let $\mathbb{F}$ be the hypothesis class of a DRL extraction attack. The instance and output spaces of $\mathbb{F}$ are denoted as $\mathbb{A} \times \mathbb{S}$ and $[0, 1]$, respectively. We refer to the sequence $S = \{s_1, s_2, ..., s_T\}$, $s_i \in \mathbb{S}$ as the environmental states of one episode, and $A = \{a_1, a_2, ..., a_T\}$, $a_i \in \mathbb{A}$ as the corresponding actions of the targeted model. The adversary's goal is to find a policy $\pi \in \mathbb{F}$ such that π performs competitively with the targeted policy π^*. Following the assumptions in the threat model where the adversary can operate the model in an environment and learn π by observing the corresponding actions of π^*. We now formally define the DRL extraction attack:

Definition 1. *((δ, ξ)-DRL Extraction Attack) Let π be the reproduced policy in a DRL extraction attack $\mathcal{A}$. We call $\mathcal{A}$ a (δ, ξ)-DRL extraction attack if for $\forall S = \{s_1, s_2, ..., s_T\}$, the difference between the expected cumulative rewards of π^* and π is smaller than δ (accuracy extraction) and the expected distance between the action distributions of π^* and π is smaller than ξ (fidelity extraction), i.e.,*

$$|E[\sum_{i=1}^{T} r(s_i, a_i^*)] - E[\sum_{i=1}^{T} r(s_i, a_i)]| \leq \delta, \tag{5}$$

$$E[d(\{a_1^*, a_2^*, ..., a_T^*\}, \{a_1, a_2, ..., a_T\})] \leq \xi, \tag{6}$$

where $\{a_1^*, a_2^*, ..., a_T^*\}$ and $\{a_1, a_2, ..., a_T\}$ are the corresponding actions of π^* and π on the sequence of states. E is the expected value of a random variable.

3.3 Imitation Learning

Imitation learning was originally introduced for learning from human demonstrations. Then its concept was applied to the domain of artificial experts, such as RL agents. Given a task, an imitation model acquires the corresponding skills from expert demonstrations by learning a mapping between observations and actions [30]. A formal definition of imitation learning is described below:

Definition 2. *(Imitation Learning System) Let $\mathbb{F}'$ be a hypothesis class with instance space $\mathbb{X}$ and output space $\mathbb{Y}$. An imitation learning system for $\mathbb{F}'$ is given by two entities $(\mathcal{A}, O)$: O is the teacher that plays as an oracle with the optimal hypothesis. $\mathcal{A}$ is the agent that searches or learns the optimal hypothesis from $\mathbb{F}'$ by observing O's behaviors $\{(x, y)\}$, $x \in \mathbb{X}, y \in \mathbb{Y}$.*

The agent $\mathcal{A}$ adopts an imitation learning algorithm $\mathcal{L}$ to learn the behaviors of the teacher O in the system. We define such an algorithm below:

Definition 3. *(ϵ-Imitation Learning Algorithm) Let $\mathcal{L}$ be an algorithm used by $\mathcal{A}$ to learn the behaviors of O. Let π be an "expert" that describes O's behaviors, and π' be the hypothesis learned by $\mathcal{L}$. We say $\mathcal{L}$ is an ϵ-imitation learning algorithm if for $\forall\{s_1, s_2, ..., s_T\}$, the expected value of the distance between the corresponding action distributions of π^* and π' is smaller than ϵ, i.e.,*

$$E[d(\{a_1^*, a_2^*, ..., a_T^*\}, \{a_1', a_2', ..., a_T'\})] \leq \epsilon, \tag{7}$$

where $d(\cdot, \cdot)$ is a measure of two probability distributions.

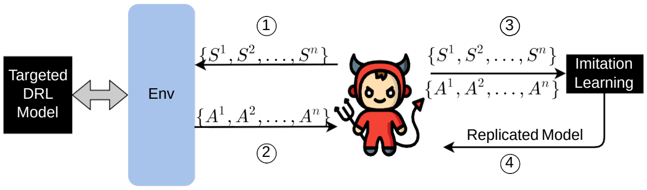

Targeted DRL Model
Env
①
$\{S^1, S^2, \ldots, S^n\}$
$\{A^1, A^2, \ldots, A^n\}$
②
③
$\{S^1, S^2, \ldots, S^n\}$
$\{A^1, A^2, \ldots, A^n\}$
Imitation Learning
Replicated Model
④

Figure 1: Constructing a (δ, ξ)-DRL extraction attack based on an ϵ_δ-imitation learning algorithm.

Intuitively, an algorithm is an ϵ-imitation learning algorithm if the action distribution of the hypothesis learned by it is similar to the corresponding teacher with the distance bounded by the threshold ϵ. In our experiments, we adopt the Jensen-Shannon divergence to measure the action distributions of two models.

3.4 Connections between DRL Model Extraction and Imitation Learning

Based on the descriptions in the previous two sections, we can observe that DRL model extraction closely resembles the DRL imitation learning scenario. In this section, we formally prove that DRL model extraction can be cast as imitation learning under the assumption that the hypothesis classes of the DRL extraction attack and imitation learning are the same:

Theorem 3.1. *Let $\mathbb{F}^*$ be the hypothesis class for searching the targeted policy π^* in an imitation learning system, $\mathbb{F}$ be the hypothesis class of a model extraction adversary with the same instance and output spaces as $\mathbb{F}^*$. We assume the adversary owns an ϵ_δ-imitation learning algorithm, and uses it to learn a policy π'. Let t_ϵ be the expected value of the number of different actions between π^* and π'. Then this adversary is able to conduct a (δ, ξ)-DRL extraction attack if (1) $\mathbb{F} = \mathbb{F}^*$, (2) the maximal reward of π' on t_ϵ actions is smaller than δ, and (3) $\xi \leq \epsilon_\delta$.*

Proof. We construct a two-step DRL extraction attack, as shown in Figure 1. At step 1, the adversary randomly selects some states $\{S^1, S^2, ..., S^n\}$, where $S^i = \{s_1^i, ..., s_T^i\}$, and feeds them to the targeted DRL model (①). He then observes the corresponding actions from this model $\{A^1, A^2, ..., A^n\}$, where $A^i = \{a_1^i, ..., a_T^i\}$ (②). At step 2, with the collected states and actions, the adversary adopts an ϵ_δ-imitation learning algorithm to learn a new policy that can mimic the actions from the states (③). The adversary can repeat the two steps for multiple times until a qualified policy is acquired, which will be the output of the model extraction attack (④).

We now prove that this two-step attack is a (δ, ξ)-DRL extraction attack. One can easily find that the attack satisfies Equation 6 because $\xi \leq \epsilon_\delta$. We show that the attack also satisfies Equation 5 as follows. Since an ϵ_δ-imitation learning algorithm is adopted, for $\forall\{s_1, s_2, ..., s_T\}$, we have

$$E[d(\{a_1^*, a_2^*, ..., a_T^*\}, \{a_1', a_2', ..., a_T'\})] \leq \epsilon_\delta. \tag{8}$$

t_ϵ is the expected number of actions differ in π^* and π'. Here, we denote $\{(s_1^{max}, a_1^{max}), ..., (s_{t_\epsilon}^{max}, a_{t_\epsilon}^{max})\}$ as the state-action pairs that obtain the maximal reward than other t_ϵ pairs, i.e.,

$$\delta_{t_\epsilon} = \sum_{i=1}^{t_\epsilon} r(s_i^{max}, a_i^{max}) \geq \sum_{i=1}^{t_\epsilon} r(s_i, a_i'). \tag{9}$$

Since $\delta_{t_\epsilon} < \delta$, we have

$$|E[\sum_{i=1}^{T} r(s_i, a_i^*)] - E[\sum_{i=1}^{T} r(s_i, a_i)]| \tag{10}$$

$$= E[\sum_{i=1}^{t_\epsilon} r(s_i, a_i')] \leq \delta_{t_\epsilon} \leq \delta, \tag{11}$$

□

4 PROPOSED ATTACK

Based on Theorem 3.1, we can adopt well-developed imitation learning techniques to extract DRL models. However, it also requires the hypothesis classes of the two problems must be the same. In another word, the adversary should use the same training algorithm of the targeted DRL model as the imitation learning algorithm. Unfortunately, we consider the adversary has no prior knowledge about the training algorithm of the DRL model. This makes it difficult to directly apply imitation learning solutions for model extraction. Due to the distinct features between various DRL algorithms, using a different imitation learning algorithm can hardly recover the model with high fidelity, as we will evaluate in Section 5.

To address the above issue, we propose a new methodology to identify the training algorithm family (i.e., the hypothesis class) of a black-box model. The key idea of this approach is that DRL models trained from different algorithms can yield different behaviors from the same states. Such temporal sequences can be distinguished by machine learning classifiers, which can leak the training algorithm to the adversary from the model's dynamics.

Hence, we present our end-to-end attack approach that consists of two stages. At the first stage, we construct a classifier, which can be used to predict the *training algorithm family* of a given black-box DRL model from its runtime behaviors. At the second stage, based on the extracted information, we adopt state-of-the-art imitation learning techniques to generate a model with similar *behaviors* as the victim one. Figure 2 illustrates the methodology overview, and Algorithm 1 describes the detailed steps.

4.1 Identifying Training Algorithm Families

We first train a classifier, whose input is a DRL model's action sequence, and output is the model's algorithm family. With this classifier, we can identify the algorithm family of an arbitrary model.

Dataset preparation. A dataset is necessary to train this classifier. It should include action sequences of DRL models trained from different algorithms. To this end, we first train a quantity of shadow DRL models in the same environment but with various algorithms, and then collect their behaviors to form this dataset.

Specifically, we set up an algorithm family pool $\mathbb{P}$ that includes all the DRL algorithm families in our consideration. We also prepare a set $\mathbb{S}$ of random seeds for environment initialization. For each algorithm family in $\mathbb{P}$, we train some DRL models in the environments initialized by different random seeds in $\mathbb{S}$. We evaluate the performance of each trained DRL model by comparing its reward with a reward threshold R: we only select the DRL models whose rewards are higher than R. For each qualified model, we collect N different state-action sequences with a length of T: $\{(s_1, a_1), (s_2, a_2), ...(s_T, a_T)\}$. Then samples are generated with the action sequences ($A = \{a_1, a_2, ...a_T\}$) as the feature and the training algorithm family as the label, to construct the dataset.

Algorithm 1: Extracting DRL models

```
Input: Targeted model $M^*$, DRL environment $env$
Output: Replicated model $M'$
/* Stage 1 */
Set up a set of random seeds $\mathbb{S}$, reward threshold $R$, action
  sequence length $T$ ;
Select algorithm family pool $\mathbb{P}$; Dataset $\mathbb{D} = \emptyset$ ;
for each $p \in \mathbb{P}$ do
    for each $s \in \mathbb{S}$ do
        $env$.initialize($s$);
        $m$ = train_DRL($env$, $p$);
        if eval($m$, $env$) > $R$ then
            $A$ = GenSequence($m$, $env$, $T$);
            $\mathbb{D}$.add([$A$, $p$]);
$C$ = train_RNN($\mathbb{D}$);
/* Extract algorithm family */
$A^*$ = GenSequence($M^*$, $env$, $T$);
$P^*$ = $C$.predict($A^*$)
/* Stage 2 */
$M'$ = ImitationLearning($M^*$, $P^*$, $env$);
while eval($M'$, $env$) < eval($M^*$, $env$) do
    $M'$ = ImitationLearning($M^*$, $P^*$, $env$);
return $M'$
```

Training. We adopt the Recurrent Neural Network (RNN) as the classifier[1]. An RNN is competent of processing sequence data of arbitrary lengths by recursively applying a transition function to its internal hidden state vector of the input. It is generally used to map the input sequence to a fixed-sized vector, which will be further fed to a softmax layer for classification. However, vanilla RNNs suffer from the gradient vanishing and exploding problem: during training, components of the gradient vector can grow or decay exponentially over long sequences. To address this problem, we adopt the Long Short-Term Memory (LSTM) network [24] in our design. LSTMs can selectively remember or forget things regulated by a set of gates. Each gate has a sigmoid neural net layer and a pointwise multiplication operation, which can filter the information through the network. Therefore, LSTM units can maintain information in memory for a long period under the control of the gates.

To train the classifier over the prepared dataset, for each input sequence $A = \{a_1, a_2, ..., a_T\}$, we first apply a set of LSTM layers to obtain its vector representation. Then we attach a fully-connected layer and a non-linear softmax layer after the LSTMs to output the

[1] In Section 5 we also implemented and tested other network structures as well. RNN gives the best results for recognizing sequential features.

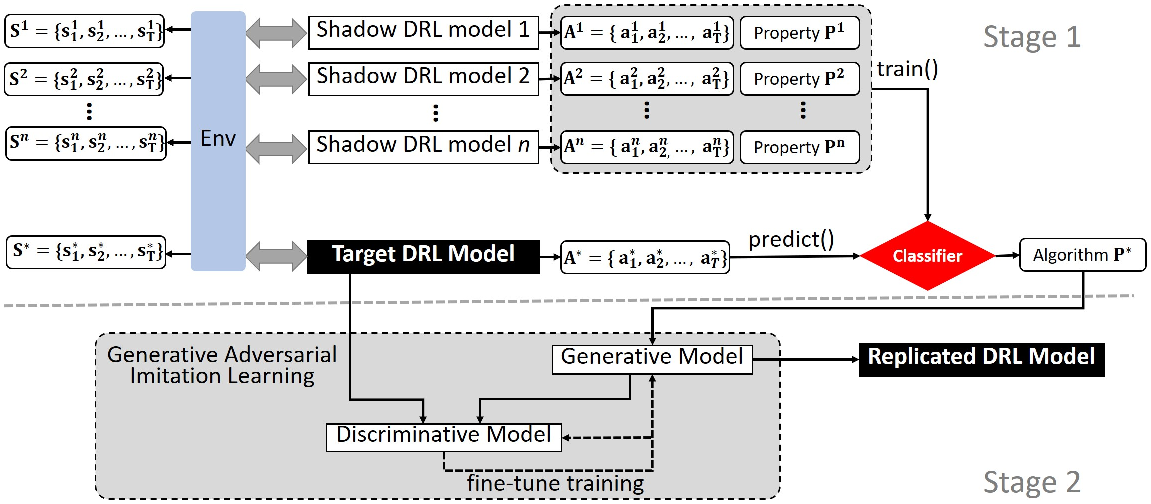

$S^1 = \{s_1^1, s_2^1, \ldots, s_T^1\}$
$S^2 = \{s_1^2, s_2^2, \ldots, s_T^2\}$
$S^n = \{s_1^n, s_2^n, \ldots, s_T^n\}$
$S^* = \{s_1^*, s_2^*, \ldots, s_T^*\}$
Env
Shadow DRL model 1
Shadow DRL model 2
Shadow DRL model n
$A^1 = \{a_1^1, a_2^1, \ldots, a_T^1\}$
$A^2 = \{a_1^2, a_2^2, \ldots, a_T^2\}$
$A^n = \{a_1^n, a_2^n, \ldots, a_T^n\}$
Property P^1
Property P^2
Property P^n
Stage 1
train()
Target DRL Model
$A^* = \{a_1^*, a_2^*, \ldots, a_T^*\}$
predict()
Classifier
Algorithm P^*
Generative Adversarial Imitation Learning
Generative Model
Replicated DRL Model
Discriminative Model
fine-tune training
Stage 2

Figure 2: Overview of our proposed DRL model extraction attack

probability distribution over all classes of possible model algorithms. We use cross-entropy loss to identify the optimal parameters for this classifier by minimizing the loss function.

Extracting algorithm families. With this RNN classifier, we are now able to predict the training algorithm family of a given backbox DRL model. We operate this targeted model in the same environment with certain random seed and collect the action sequence for T rounds. We query the classifier with this sequence and get the probability of each candidate algorithm family. We select the one with the highest probability as the attack result. To further increase the confidence and eliminate the stochastic effects, we can run the targeted model in different initialized environments and collect the sequences for predictions. We choose the most-predicted label as the targeted model's training algorithm family.

4.2 Extracting Models via Imitation Learning

With the extracted training algorithm family, the adversary can now perform a (δ, ξ)-DRL extraction attack via imitation learning. Various techniques have been designed to imitate the behaviors of DRL models. For instance, inverse optimal control was adopted in [15] to learn the behaviors from expert's demonstrations. Inverse Reinforcement Learning (IRL) [47] was then proposed to improve the learning effects when no clear reward function is available. Generative Adversarial Imitation Learning (GAIL) [23] was proposed to directly learn policies using Generative Adversarial Framework. Deep Q-learning from Demonstration (DQfD) [22] combined temporal difference updates with supervised classification of the expert's actions to accelerate the training of DRL models.

In our implementation, we adopt the GAIL framework for model extraction, as it can be easily converted to an ϵ_δ-imitation learning algorithm. GAIL is a model-free method that can obtain significant performance gains in imitating complex behaviors in large-scale and high-dimensional environments. It generalizes Inverse Reinforcement Learning [47] by formulating the imitation learning as minimax optimization, which can be solved by alternating gradient-type algorithms in a more scalable and efficient manner.

Specifically, similar as Generative Adversarial Networks, GAIL is composed of two neural networks, which contest with each other during the imitation process: a generative DRL model G, and a discriminative model D whose job is to distinguish the distribution of data generated by the generator G from the ground-truth data distribution of the expert DRL model. Given a sequence of states S_i and the corresponding actions A_i, D outputs the probability of the sequences generated by the expert model. During the training process, D is optimized to accurately distinguish the state-action sequences generated by G and the expert model. In particular, GAIL updates D iteratively using the gradient ascent technique to minimize the loss function:

$$L_D = -\mathbb{E}_G[\log(D(S_i, A_i))] - \mathbb{E}_{\pi^*}[\log(1 - D(S_i, A_i^*))], \quad (12)$$

where S_i, A_i are generated by G, while S_i, A_i^* are produced by the expert model π^*.

A satisfactory generator G should behave similar as the expert model. To this end, GAIL requires G to increase the probabilities of D on the sequences it generates and optimize G by minimizing the following loss function,

$$L_G = -\sum_{i=1}^{n} D(S_i, A_i) P(S_i, A_i | G), \quad (13)$$

where $P(S_i, A_i|G)$ is the probability that the sequences S_i, A_i occur given the generator G.

Then the overall optimization goal of GAIL is to minimize the Jensen-Shannon divergence between the behaviors of G and the expert model [23]:

$$\arg\min_G \arg\max_D L(G, D) = \arg\min_G d_{JS}(\rho_G || \rho_{\pi^*}) - R(G) \quad (14)$$

where $R(G)$ is the policy regularizer and $d_{JS}(p||q) = d_{KL}(p||(p+q)/2) + d_{KL}(q||(p+q)/2)$ represents the Jensen-Shannon divergence between two distributions p and q. ρ_{π^*} is a joint distribution of states and actions following π. During the learning process, GAIL

Algorithm	RNN	MLP	SVM	Random Forest
A2C	82%	62.4%	74.8%	34.8%
PPO	99%	79.4%	79.6%	77.8%
ACER	54%	41.8%	27.6%	28.6%
ACKTR	82%	67.6%	28.6%	44.8%
DQN	95%	47%	21%	37.6%
Average	82.4%	60%	46.3%	44.6%

Table 1: The accuracy of different classifiers

alternately trains the generative and discriminative models until $d_{JS}(\rho_G||\rho_{\pi^*})$ is smaller than ϵ_δ.

Considering the stochasticity of the imitation learning process, it is possible that the model cannot reach the same reward although it has the same behaviors as the targeted model. Therefore, we repeat the GAIL process until a qualified model is obtained which has very similar performance (i.e., reward) as the targeted model.

5 EVALUATION

5.1 Implementation and Experimental Setup

Our attack approach is general and applicable to various reinforcement learning tasks. Without loss of generality, we consider two popular environments: Cart-Pole and Atari Pong [13]. For each environment, we train DRL models with five mainstream DRL algorithm families (DQN [40], PPO [48], ACER [56], ACKTR [58] and A2C [39]). We believe these two tasks with five algorithms are representative, as they are commonly used in academia for evaluating reinforcement learning applications. These algorithms also demonstrate great potential in real-world products. All the model configurations and hyperparameters follow the default setup in the OpenAI Baselines framework [13].

For the algorithm classification, we select 50 shadow models from each DRL algorithm and environment whose rewards are higher than a task-specific threshold R, defined in the OpenAI framework. We consider different sequence lengths T (50, 100 and 200), and compare their impacts on the prediction accuracy. For each shadow DRL model, we collect 50 action sequences as the training inputs of our classifier. Therefore, for Cart-Pole and Atari Pong, the sizes of the datasets from 250 shadow models are both 12,500. To evaluate the classifier, we randomly split the trajectory data to training and test sets with a ratio 8:2. During the training process, the initial learning rate is set to 0.005 with a decay factor of 0.7 when loss plateaus occur. The batch size is set to 32. Without loss of generality, we set $\xi = \epsilon_\delta$. Instead of predefining ϵ and the corresponding ϵ_δ for each task, we stop the training after $N = 100$ iterations in.

5.2 Results of Algorithm Family Identification

Comparisons of different classification models. As the first study, we investigate the effectiveness of different machine learning models on predicting DRL training algorithm families. We implement the classifiers based on various common machine learning models, i.e., RNN, MLP, SVM and random forest. Their prediction accuracy for each algorithm family as well as the average values are shown in Table 1. We observe the RNN classifier achieves the highest accuracy, followed by the MLP classifier. The performance of SVM and random forest is extremely low, since they are not good at handing sequential inputs. Therefore, RNN is the optimal choice for classification, and will be used for the rest experiments.

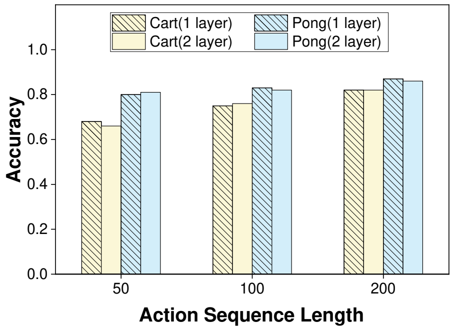


Figure 3: Average accuracy of RNN

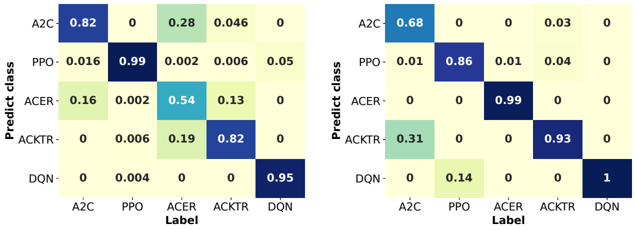
Impact of hyperparameters for the algorithm classifier. The accuracy of the RNN classifier can be affected by a few hyperparameters, e.g., the length of the input sequence, the number of hidden layers. Figure 3 shows the accuracy under different combinations of these settings. First, we observe that the length of the input sequence can affect the classification performance: a longer input sequence can give a higher accuracy. Therefore, for Cart-Pole environment, we take all the actions within one episode as the input sequence (T=200). For Atari Pong environment, one episode can have up to 10,000 actions. It is not recommended to take the entire episode as input, which can incur very high cost and training over-fitting. Since $T = 200$ can already give us very satisfactory accuracy, we will set the length of input sequence to 200 as well.

Second, we consider different numbers of hidden LSTM layers (1 and 2) for the classifier. We observe that this factor has slight influence on the accuracy of the classifier. One hidden layer can already validate the effectiveness of the RNN classification. So in the following experiments, we will adopt a 1-layer RNN for simplicity.

Third, the action space can also affect the classification accuracy. Higher-dimensional actions contain more information about the DRL model, and can be identified more accurately. In our case, the action space of Cart-Pole environment is 2 while that of Atari Pong environment is 6. Then the classification of Atari Pong has a higher accuracy than Cart-Pole, as reflected in Figure 3.

Accuracy of each class. Figures 4 shows the confusion matrix for both environments. We observe the classifier can distinguish DRL models of each algorithm family with very high confidence. For most cases, the prediction accuracy is above 70%; the best case is 100% (DQN models in Atari Pong); the worst case is 54% (ACER models in Cart-Pole), which is still much higher than random guess (20%). The prediction accuracy of the DQN model is particularly high (95% in Cart-Pole, and 100% in Atari Pong). This is because DQN is a value-based algorithm while all the other algorithms are actor-critic methods. So DQN models are easier to be distinguished.

Impact of hyperparameters and structures for DRL models. When training the classifier, we use the default hyperparameters and model structures in the OpenAI Baselines benchmark [13] to



(a) Cart-Pole

(b) Atari Pong

Figure 4: The accuracy of identified algorithm families

Algorithm	Default	Learning Rate		Train step	Structure
		+50%	-20%	+100%	+1 Layer
A2C	82%	73%	86%	71%	72%
PPO	99%	98%	91%	83%	65%
ACER	54%	58%	52%	71%	59%
ACKTR	82%	86%	91%	87%	66%
DQN	95%	80%	72%	88%	96%
Average	82.4%	79%	78.4%	80%	71.6%

Table 2: The impact of hyperparameters and network structures in the same family

generate shadow DRL models and collect the sequence features. However, our classifier is still able to predict the algorithm family of a DRL model which uses a different set of hyperparameters and network structures, as the training algorithm dominates the network configurations in reflecting the model's characteristics.

To confirm this, we train targeted DRL models with different hyperparameter values and model structures, and use our classifier to predict their algorithms. Specifically, considering the unstable learning process of large hyperparameters variance, we alter the learning rates (i.e., 150% and 80% of the default one) and training steps (double of the default one) properly in the Cart-Pole model training progress. The default network structures of all the algorithm families in our consideration consist of two hidden layers with 64 neurons, with an exception of DQN which has only one hidden layer. We modify the model structures by adding one more hidden layer to each network. The prediction results are shown in Table 2. We observe that the prediction accuracy is still very high when the hyperparameters and structures are different from what are used during training. The results indicate that our classifier can learn the characteristics of different algorithm families instead of just memorizing certain samples.

Interpretation of the classification results. We quantitatively explain why our classifier can distinguish different DRL algorithm families. We adopt the Local Interpretable Model-agnostic Explanations (LIME) framework [46], which understands a model by perturbing its input and observing how the output changes. Specifically, it modifies a single data sample by tweaking the feature values and observes the resulting impact on the output to determine which features play an important role in the model predictions.

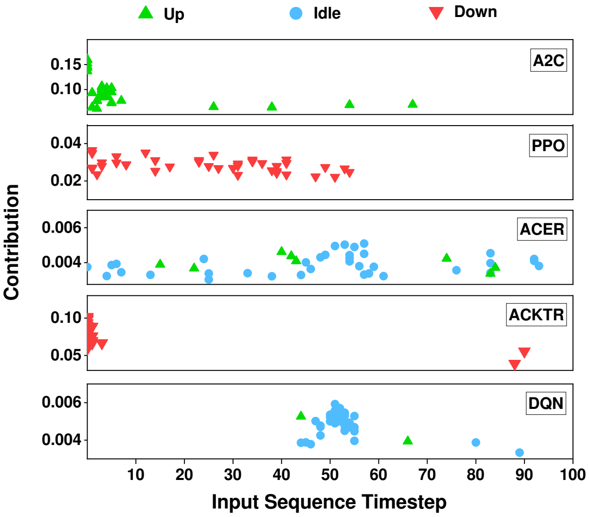


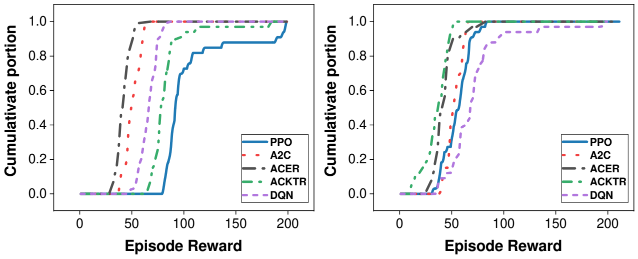
Figure 5: Action preference of different algorithm families

In our case, we build an explainer with LIME on our RNN classifier. Then, we randomly select 200 explanation instances from the training data of the classifier in Atari Pong environment. We feed these instances to the explainer and obtain the explanation results. For each explanation instance, we identify the feature (i.e., one action of the input sequence) which contributes most to the prediction. Through the analysis of these features, we can discover the different behaviors of DRL models trained from different algorithms. Figure 5 shows the contribution of the actions (UP, DOWN, IDLE) with prominent impacts on the prediction in each input sequence. We can observe that models trained with different DRL algorithm families give very different behavior preferences. A2C tends to issue important actions of UP at the beginning of the sequence; ACKTR prefers to give the action of DOWN also at the beginning of the task; DQN has a higher chance to predict IDLE clustering at the beginning; PPO issues the DOWN action all over the sequence with a large variance of contribution factor; ACER has important actions of UP and IDLE with similar contributions spanning all over the sequence. This shows those DRL algorithms have quite different characteristics in making action decisions, giving the classifier an opportunity to distinguish them just based on the actions.

5.3 Results of Imitation Learning

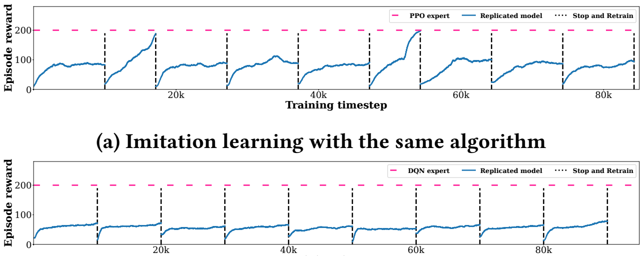
We demonstrate the effectiveness of imitation learning for model extraction. To train the replicated models, GAIL is applied to imitate the behaviors of the targeted model. We use two different types of DRL algorithms (i.e., value-based algorithm DQN and actor-critic algorithm PPO) as the generators of GAIL. Other policies can be applied in the same way. We follow [13, 34] to implement the PPO and DQN generators of GAIL, respectively. Without loss of generality, we evaluate the imitation effects in Cart-Pole environment.

Accuracy extraction. The extracted model with the same algorithm family can reach similar performance (i.e., reward) as the targeted model after imitation learning. We assume the targeted



(a) Performance accuracy (PPO) (b) Performance accuracy (DQN)

Figure 6: Results of accuracy extraction



(a) Imitation learning with the same algorithm

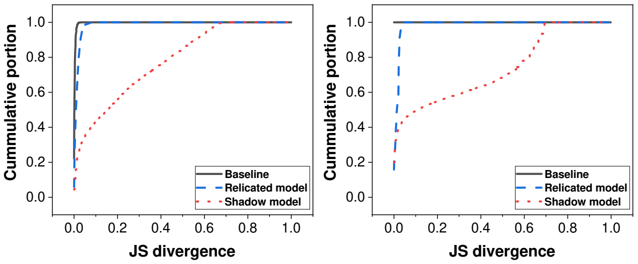
(b) Imitation learning with different algorithms

Figure 7: The reward curve of the extracted model during the imitation learning process

model is in the PPO and DQN family, and the adversary has extracted such information via RNN classification. Then he adopts the GAIL framework to learn a new model from the same family. We repeat this process 30 times, and Figure 6 shows the cumulative probability of the episode reward of all extracted models (blue solid line in Figure 6a and purple dash line in Figure 6b). We can observe that the extracted model has a big chance to reach high episode rewards as the targeted model. In contrast, when the adversary uses a different training family from the targeted model for imitation learning, the extracted model can rarely reach high rewards. This indicates the importance of the algorithm identified by the RNN classification, in order to perform high-quality imitation learning.

To further study the performance of the extracted models imitated with the same algorithm as the targeted model, we demonstrate the learning process (both models adopt PPO) in Figure 7a. We observe that in the first imitation cycle, the extracted model cannot reach the same reward as the targeted one, as it learns the random behaviors of the targeted model with low rewards. Then we start a new imitation cycle, and now the learned model can get the same reward as the victim model (i.e., 100%). We can stop with this replica, or continue to identify more qualified ones (at the 6th cycle). In contrast, we also consider a case where the adversary does not know the training algorithm, and randomly pick one for imitation learning. Figure 7b shows the corresponding imitation process (the targeted model uses ACER while the adversary selects the PPO generator). Now the extracted model can never get the same reward as the targeted model.

Fidelity extraction. In addition to the rewards, the extracted model can also learn similar behaviors as the targeted one. Since the output of a DRL model is a probability distribution over legal actions, we adopt the Jensen-Shannon (JS) divergence [36] to measure the similarity of the action probability distributions between the imitated and the targeted models. We consider three cases: (1) the similarity between the targeted model and itself (i.e., collecting the behaviors twice). This serves as the baseline for comparison. (2) the similarity between the targeted model and the extracted model from imitation learning; (3) the similarity between the targeted model and a shadow model which is trained from scratch with the same algorithm rather than imitation learning. For each case, we feed the same states to the two models in comparison, sample 100 actions from each model, compute the action probability distributions and the divergence between these two action distributions. Figure 8 shows the cumulative probability of the JS divergence for each case. For the PPO algorithm, we can observe that the cumulative portion of JS divergence in both cases (1) and (2) increases sharply to 1 (the JS divergence on 97% states is smaller than 0.05). Since the DQN algorithm is a deterministic method, which always outputs the same optimal action, all the behavior divergence in case (1) is zero and the JS divergence in case (2) also increases sharply to 1. This indicates that the extracted model indeed has very similar behaviors as the targeted model.



(a) Behavior fidelity (PPO) (b) Behavior fidelity (DQN)

Figure 8: Results of fidelity extraction

In contrast, the divergence of action probability distributions between the shadow model and targeted model can be very high. Even they are trained from the same algorithm family, their behaviors are still quite distinct in the same environment. We can conclude that through imitation learning with the identified algorithm family, the extracted model can behave very closely with the targeted one.

6 CASE STUDY

To demonstrate the effectiveness and severity of our attack methodology, we present two attack cases. The first case is to utilize model extraction to increase the transferability of adversarial examples against black-box DRL models. The second case is to steal a proprietary model and remove the watermarks embedded in it.

6.1 Enhancing Adversarial Attacks

As the first case, we show how we can leverage model extraction to facilitate existing adversarial attacks against DRL models.

Motivation. One of the most severe security threats against machine learning models is the adversarial example [52]: with imperceptible and human unnoticeable modifications to the input, a machine learning model can be fooled to give wrong prediction

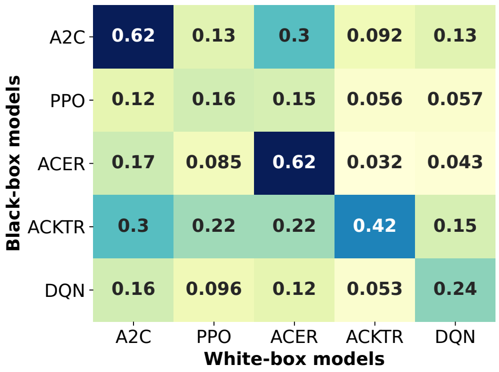


Figure 9: Enhancing transferability

results. Following this finding, many researchers have designed various methods to attack supervised DL models [8, 17, 45]. Adversarial attacks against RL policies have also received attention over the past years. Huang et al. [28] made an initial attempt to attack neural network policies by applying the conventional method (FGSM) to the state at each time step. Enhanced adversarial attacks against DRL policies were further demonstrated for higher efficiency and success rates [6, 59].

Adversarial attacks enjoy one important feature: *transferability* [52, 60]. Malicious samples generated from one model has certain probability to fool other models as well. Due to this transferability, it becomes feasible for an adversary to attack black-box models, as he can generate adversarial examples from an alternative white-box model and then transfer them to the targeted model.

The transferability of adversarial examples depends on the similarity of the white-box and black-box models. The adversarial examples have higher chance to fool the black-box model which shares similar features and behaviors as the white-box one. This becomes difficult in the scenario of DRL due to the models' complexity and large diversity, especially when they are trained from different algorithms. Hence, we can leverage the model extraction technique proposed in this paper to improve the transferability. Specifically, we identify the training algorithm from the black-box DRL model and replicate a new one. Then we generate adversarial examples via conventional methods from the parameters of the extracted model, and use them to attack the targeted black-box one.

Implementation. We evaluate the effectiveness of our improved adversarial attacks in Atari Pong. We conduct adversarial attacks on five commonly used DRL algorithm families (i.e., A2C, PPO, ACKTR, ACER, DQN). Each experiment is repeated 3 times and the average results are reported. For each targeted model, we also choose other different DRL algorithms to train shadow models as the baseline. For each white-box shadow model, we generate 1,000 adversarial examples by utilizing the FGSM technique [17],

$$s^{adv} = s + \epsilon sign(\nabla_s L(s, a)), \quad (15)$$

where ϵ represents the perturbation magnitude set as 0.15 in our experiments. We attack the targeted models with the generated adversarial examples and collect their success rates.

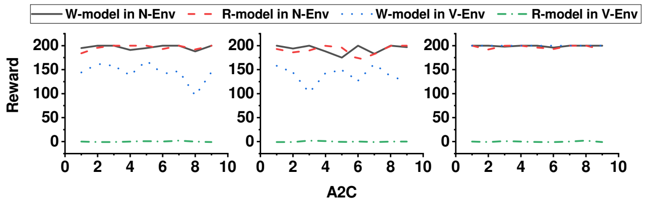


Figure 10: The performance of watermarked and extracted models in the normal and verification environments

Results. Figure 9 reports the success rates (transferability) of adversarial examples across different algorithms under the same perturbation scale. We observe that the success rate increases when the extracted model is trained by the same training algorithm family as the targeted model. The reason is that because the gradients of the DRL models with the same algorithm family are closer than the others, adversarial examples are easier to be transferred. This indicates that our model extraction technique can significantly enhance the black-box adversarial attacks. It is worth noting that since PPO is more robust against adversarial attacks [28], the success rates on PPO is generally lower.

6.2 Invalidating Watermark Protection

Motivation. DL models have become important Intellectual Property (IP) for AI companies and practitioners. An efficient way for IP protection is watermarking. This technique enables the ownership verification of DL models by embedding watermarks into the models. Then the owners can extract the watermarks from suspicious models as the evidence of ownership. Various techniques have been designed to watermark DNN models [3, 54]. In the DRL scenario, the most common approach is to embed a sequential pattern of states and behaviors into the targeted DRL model as watermarks [5, 11]. In this case study, we show that even a DRL model is protected by this watermark mechanism, our extraction technique is able to replicate the model and remove the embedded watermarks. In another word, our solution is able to learn the behaviors of the targeted model while ignoring the hidden watermarks.

Implementation. We consider three DRL algorithm families (A2C, PPO, DQN) with the Cart-Pole environment and train the corresponding watermarked DRL models following the implementation in [5]. Specifically, we first create five out-of-distribution watermark. At the i-th state, we define the corresponding action and the next state as $i\%2$ and state$[i\%4 + 1]$, respectively. To embed the watermarks, we reset the reward of the predefined action on the i-th state to 1. If the DRL model acts differently from the watermark actions, we compulsorily terminate the episode and return a reward of -1. During the DRL training and watermark embedding process, we train DRL models in the new verification environment until the watermarks are successfully embedded and the watermarked models can achieve high reward values (i.e., 195). For each DRL algorithm family, we train three watermarked models with the above scheme and report the average results.

Results. Figure 10 shows the performance of the targeted watermarked model, as well as the extracted model using our proposed technique. The x-axis and y-axis represent the index of the experiment and the corresponding reward, respectively. In each figure, we show four lines: the watermarked model in the normal environment (W-model in N-Env) and verification environment (W-model in V-Env); the replicated model in the normal environment (R-model in N-Env) and verification environment (R-model in V-Env).

We observe that, in the normal environment, the performance of the watermarked model and the replicated model is identical, indicating the success of model extraction. In contrast, in the verification environment, the replicated model has quite different behaviors and reward from the watermarked model. We conclude that the model extraction attack will not learn the watermarks from the targeted model, for all the three algorithms. We also observe that the watermarked model in the case of DQN outperforms other algorithms in the verification environment. This is because the DQN algorithm is deterministic and always outputs the optimal actions, while PPO and A2C are stochastic that sample actions following their action probabilities. Thus, if a watermarked model samples an action different from the predefined action, PPO and A2C will terminate the process and get a lower reward.

7 DISCUSSION

7.1 Attack Complexity

The cost of our attack originates from algorithm classification and imitation learning. For the first step, the main effort is to train a number of DRL models for data collection, and train an algorithm classifier. Note that this offline step incurs only one-time cost. After the classifier is produced, we can use it to attack arbitrary DRL models for infinite times. This makes the training effort acceptable. Future works will focus on reducing the number of shadow models to maintain or improve the classification performance. The online prediction process is very fast with negligible overhead.

For the second step, the adversary needs to learn a model via imitation learning. This is comparable to training a new model from scratch. Sometimes the adversary may not obtain an ideal model with the same reward due to the high stochasticity in the DRL learning process. He has to repeat this step until a qualified model is identified. From our empirical results, the adversary can find the optimal model in a couple of rounds with very high probability if he uses the correct training algorithm. Besides, the adversary can perform fine-tuning over a well-trained model (e.g., a shadow model from the same training family at step 1) to speed up this imitation learning process.

7.2 Extension to More Complex Scenarios

Our attack method follows the general modeling of DRL scenarios to predict and extract the targeted model via the observed actions. In our experiments, we evaluate the proposed attack on two standard DRL benchmark tasks (i.e., Cart-Pole and Atari Pong). It is expected to also work on other more complex DRL applications such as robotics navigation [66] and motion control [35].

Our approach can be extended to other algorithms as well. We can train a classifier to recognize the features of new algorithms in a similar way. Particularly, some reinforcement learning algorithms may have states and actions that are not fully observable (e.g., partially observable Markov Decision Process [9]). In this case, the adversary can just select the observable state-action pairs while ignoring the hidden ones. Then he can follow our attack technique to perform model extraction.

Another complex case is that the training algorithm of the targeted model may not fall into the candidate pool. For instance, the model can be an ensemble of multiple DRL and CNN models; the task is modeled as a multi-agent reinforcement learning problem. In this case, our classifier is not able to predict the algorithms correctly. To the best of our knowledge, extracting a combination of multiple models is an open problem and has not been solved yet. This will be an promising future direction in our consideration.

7.3 Potential Defenses

As discussed in Section 8.2, existing defenses mainly focus on extraction of supervised DL models. It is hard to apply them to protect DRL models. Hence, a defense solution specifically designed for our attack is required. We propose three possible ideas. Validation and implementation of these solutions will be our future work.

The first possible defense is to distinguish the query behaviors of imitation learning from normal operations. Imitation learning may exhibit specific query patterns, which can indicate the occurrence of model extraction. The second defense is to add higher stochasticity to the model's behaviors, making it harder for the adversary to recognize the algorithm or mimic exact behaviors. There can be a trade-off between the model usability and security that needs to be balanced. The third defense idea is to design new DRL training algorithms totally different from the ones in the family pool. If these algorithms are kept secret, the adversary is not able to perform accurate imitation learning.

8 RELATED WORK

8.1 DNN Model Extraction Attacks

Safety and privacy have became a major security concern in deep learning models [18, 29, 51, 61, 64]. A quantity of works have been done to disclose the privacy threats about the training data [16, 20, 21] or models [42, 53, 55, 62] in the deep learning development pipeline. In this paper, we mainly focus on the model privacy issue.

Model extraction attacks aim to steal a DNN model (i.e., acquiring the attributes or parameters of the model, or a good approximation) with only black-box access. These attacks can be classified into two categories. The first category is query-based attacks: the adversary treats the targeted model as an oracle, and queries it with carefully-crafted samples. The model is replicated based on the corresponding responses. Tramer et al. [53] realized this attack against modern machine learning services. Wang and Gong [55] adopted similar techniques to steal the hyper-parameters in model training. Oh et al. [42] extracted the network structure and configurations of the targeted model (e.g., number of layers, optimization algorithm, and activation function).

Advanced attacks were proposed to improve the efficiency of model extraction with fewer queries, like adversarial example-based query [62], learning-based query [31], gradients query [38]. Model extraction attacks leveraging different types of query data were also demonstrated in [12, 43, 44]. Chandrasekaran et al. [10] showed the process of model extraction is similar to active learning, which can further enhance the attack efficiency. Carlini et al. [7] treated the model extraction as a cryptanalytic problem, and applied the differential attack technique to solve this issue.

The second category is side-channel-based attacks. The adversary collects side-channel information emitted during the inference phase to infer the properties of the neural network models. This includes timing channels [14], power side channels [4], and micro-architectural side channels [25, 27].

All of the above extraction works focus on conventional neural networks. In contrast, this paper presents the first attack against DRL models. Due to the distinct features and complex mechanisms of DRL models, it is hard to directly apply the prior techniques to address this problem. We propose a novel attack method integrating algorithm prediction and imitation learning to achieve DRL model extraction with high fidelity and efficiency.

8.2 Defenses against Model Extraction

Past works introduced different defenses to enhance the privacy of training data. Meanwhile, a couple of solutions were also proposed to thwart model extraction attacks, which can be classified tino the following three categories.

The first type of solution is prediction modification. Inspired by the information theory, these solutions limit the information gained per query from the adversary to increase the difficulty of model extraction, including perturbing the output probability [10, 53], removing the probabilities for some classes [53], returning only the class output [10, 53]. The second type is to detect extraction adversaries from benign users based on query pattern analysis [32, 33]. The third type of protection is to embed watermarks into the targeted models, and identify the extracted model copy by verifying the ownership [37, 54, 63].

Unfortunately, these solutions cannot defeat our DRL model extraction attacks. The adversary just uses normal states to query the targeted model, and observes the output actions instead of the confidence scores. As such, it is hard to prevent such malicious behaviors via prediction modification or query pattern analysis. Moreover, embedded watermarks are not robust and can be easily removed by an adversary [19]. We also show that our extracted model does not contain the watermarks anymore, thus invalidating the digital watermarking protection.

9 CONCLUSION

In this paper, we design a novel attack methodology to steal DRL models. We draw the insight that DRL model extraction can be analyzed as an imitation learning process. Hence, we propose to leverage state-of-the-art imitation learning techniques (e.g., Generative Adversarial Imitation Learning) to perform the attacks. To improve the extraction fidelity, we propose to build a classifier to predict the training algorithm family of the targeted model with only black-box access. The integration of algorithm prediction and imitation learning can achieve DRL model extraction with high accuracy and fidelity. Our attack technique can be used to enhance adversarial attacks and invalidate watermarking mechanisms. We expect this study can inspire people's awareness about the severity of DRL model privacy issue, and come up with better solutions to mitigate such model extraction attack.

10 ACKNOWLEDGEMENT

We thank the anonymous reviewers for their valuable comments. This work was supported in part by Singapore National Research Foundation, under its National Cybersecurity R&D Program No. NRF2018NCR-NCR005-0001, Singapore National Research Foundation under NCR No. NRF2018NCR-NSOE003-0001, NRF Investigatorship No. NRFI06-2020-0022, Singapore Ministry of Education AcRF Tier 1 RG108/19 (S) and RS02/19, Singapore Ministry of Education (MOE) Academic Research Fund (AcRF) Tier 2 Grant No. MOE-T2EP20120-0004, and National Research Foundation, Singapore under its AI Singapore Programme No. AISG2-RP-2020-019.

REFERENCES

[1] 2020. Safe, multi-agent, reinforcement learning for autonomous driving. https://www.mobileye.com/our-technology/driving-policy/. Accessed: 2020-12-05.

[2] JUNE 2018. Learning to drive in a day. https://wayve.ai/blog/learning-to-drive-in-a-day-with-reinforcement-learning. Accessed: 2020-12-05.

[3] Yossi Adi, Carsten Baum, Moustapha Cisse, Benny Pinkas, and Joseph Keshet. 2018. Turning your weakness into a strength: Watermarking deep neural networks by backdooring. In *USENIX Security Symposium*. 1615–1631.

[4] Lejla Batina, Shivam Bhasin, Dirmanto Jap, and Stjepan Picek. 2019. CSI NN: Reverse Engineering of Neural Network Architectures Through Electromagnetic Side Channel. In *USENIX Security Symposium*. 515–532.

[5] Vahid Behzadan and William Hsu. 2019. Sequential triggers for watermarking of deep reinforcement learning policies. *arXiv preprint arXiv:1906.01126* (2019).

[6] Vahid Behzadan and Arslan Munir. 2017. Vulnerability of deep reinforcement learning to policy induction attacks. In *International Conference on Machine Learning and Data Mining in Pattern Recognition*. 262–275.

[7] Nicholas Carlini, Matthew Jagielski, and Ilya Mironov. 2020. Cryptanalytic Extraction of Neural Network Models. *arXiv preprint arXiv:2003.04884* (2020).

[8] Nicholas Carlini and David Wagner. 2017. Towards evaluating the robustness of neural networks. In *IEEE Symposium on Security and Privacy*. 39–57.

[9] Anthony R Cassandra. 1998. A survey of POMDP applications. In *Working notes of AAAI 1998 Fall Symposium on Planning with Partially Observable Markov Decision Processes*, Vol. 1724.

[10] Varun Chandrasekaran, Kamalika Chaudhuri, Irene Giacomelli, Somesh Jha, and Songbai Yan. 2018. Exploring connections between active learning and model extraction. *arXiv preprint arXiv:1811.02054* (2018).

[11] Kangjie Chen, Shangwei Guo, Tianwei Zhang, Shuxin Li, and Yang Liu. 2021. Temporal Watermarks for Deep Reinforcement Learning Models. (2021).

[12] Jacson Rodrigues Correia-Silva, Rodrigo F Berriel, Claudine Badue, Alberto F de Souza, and Thiago Oliveira-Santos. 2018. Copycat CNN: Stealing knowledge by persuading confession with random non-labeled data. In *International Joint Conference on Neural Networks*. 1–8.

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

[14] Vasisht Duddu, Debasis Samanta, D Vijay Rao, and Valentina E Balas. 2018. Stealing neural networks via timing side channels. *arXiv preprint arXiv:1812.11720* (2018).

[15] Chelsea Finn, Sergey Levine, and Pieter Abbeel. 2016. Guided cost learning: Deep inverse optimal control via policy optimization. In *International Conference on Machine Learning*. 49–58.

[16] Karan Ganju, Qi Wang, Wei Yang, Carl A Gunter, and Nikita Borisov. 2018. Property inference attacks on fully connected neural networks using permutation invariant representations. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 619–633.

[17] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. 2014. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572* (2014).

[18] Qing Guo, Felix Juefei-Xu, Xiaofei Xie, Lei Ma, Jian Wang, Bing Yu, Wei Feng, and Yang Liu. 2020. Watch out! Motion is Blurring the Vision of Your Deep Neural Networks. In *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin (Eds.), Vol. 33. Curran Associates, Inc., 975–985. https://proceedings.neurips.cc/paper/2020/file/0a73de68f10e15626eb98701ecf03adb-Paper.pdf

[19] Shangwei Guo, Tianwei Zhang, Han Qiu, Yi Zeng, Tao Xiang, and Yang Liu. 2020. The Hidden Vulnerability of Watermarking for Deep Neural Networks. *arXiv preprint arXiv:2009.08697* (2020).

[20] Zecheng He, Tianwei Zhang, and Ruby B Lee. 2019. Model inversion attacks against collaborative inference. In *Proceedings of the 35th Annual Computer Security Applications Conference*. 148–162.

[21] Zecheng He, Tianwei Zhang, and Ruby B Lee. 2020. Attacking and Protecting Data Privacy in Edge-Cloud Collaborative Inference Systems. *IEEE Internet of Things Journal* (2020).

[22] Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Dan Horgan, John Quan, Andrew Sendonaris, Ian Osband, et al. 2018. Deep q-learning from demonstrations. In *AAAI Conference on Artificial Intelligence*.

[23] Jonathan Ho and Stefano Ermon. 2016. Generative adversarial imitation learning. In *Advances in Neural Information Processing Systems*. 4565–4573.

[24] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[25] Sanghyun Hong, Michael Davinroy, Yiğitcan Kaya, Stuart Nevans Locke, Ian Rackow, Kevin Kulda, Dana Dachman-Soled, and Tudor Dumitraş. 2018. Security analysis of deep neural networks operating in the presence of cache side-channel attacks. *arXiv preprint arXiv:1810.03487* (2018).

[26] Xing Hu, Ling Liang, Shuangchen Li, Lei Deng, Pengfei Zuo, Yu Ji, Xinfeng Xie, Yufei Ding, Chang Liu, Timothy Sherwood, et al. 2020. DeepSniffer: A DNN Model Extraction Framework Based on Learning Architectural Hints. In *International Conference on Architectural Support for Programming Languages and Operating Systems*. 385–399.

[27] Weizhe Hua, Zhiru Zhang, and G Edward Suh. 2018. Reverse engineering convolutional neural networks through side-channel information leaks. In *ACM/ESDA/IEEE Design Automation Conference*. 1–6.

[28] Sandy Huang, Nicolas Papernot, Ian Goodfellow, Yan Duan, and Pieter Abbeel. 2017. Adversarial attacks on neural network policies. *arXiv preprint arXiv:1702.02284* (2017).

[29] Xiaowei Huang, Daniel Kroening, Wenjie Ruan, James Sharp, Youcheng Sun, Emese Thamo, Min Wu, and Xinping Yi. 2020. A survey of safety and trustworthiness of deep neural networks: Verification, testing, adversarial attack and defence, and interpretability. *Computer Science Review* 37 (2020), 100270.

[30] Ahmed Hussein, Mohamed Medhat Gaber, Eyad Elyan, and Chrisina Jayne. 2017. Imitation learning: A survey of learning methods. *Comput. Surveys* (2017).

[31] Matthew Jagielski, Nicholas Carlini, David Berthelot, Alex Kurakin, and Nicolas Papernot. 2020. High accuracy and high fidelity extraction of neural networks. In *USENIX Security Symposium*.

[32] Mika Juuti, Sebastian Szyller, Samuel Marchal, and N Asokan. 2019. PRADA: protecting against DNN model stealing attacks. In *IEEE European Symposium on Security and Privacy*. 512–527.

[33] Manish Kesarwani, Bhaskar Mukhoty, Vijay Arya, and Sameep Mehta. 2018. Model extraction warning in MLaaS paradigm. In *Annual Computer Security Applications Conference*. 371–380.

[34] James Lennon. 2019. *Modeling Human Behavior in Space Invaders*. Ph.D. Dissertation. Harvard College Cambridge, Massachusetts.

[35] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2015. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971* (2015).

[36] Jianhua Lin. 1991. Divergence measures based on the Shannon entropy. *IEEE Transactions on Information theory* 37, 1 (1991), 145–151.

[37] Xiaoxuan Lou, Shangwei Guo, Tianwei Zhang, Yinqian Zhang, and Yang Liu. 2021. When NAS Meets Watermarking: Ownership Verification of DNN Models via Cache Side Channels. *arXiv preprint arXiv:2102.03523* (2021).

[38] Smitha Milli, Ludwig Schmidt, Anca D Dragan, and Moritz Hardt. 2019. Model reconstruction from model explanations. In *The Conference on Fairness, Accountability, and Transparency*. 1–9.

[39] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*. 1928–1937.

[40] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529–533.

[41] Andrew Y Ng, Adam Coates, Mark Diel, Varun Ganapathi, Jamie Schulte, Ben Tse, Eric Berger, and Eric Liang. 2006. Autonomous inverted helicopter flight via reinforcement learning. In *Experimental Robotics IX*. 363–372.

[42] Seong Joon Oh, Bernt Schiele, and Mario Fritz. 2019. Towards reverse-engineering black-box neural networks. In *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*. 121–144.

[43] Tribhuvanesh Orekondy, Bernt Schiele, and Mario Fritz. 2019. Knockoff nets: Stealing functionality of black-box models. In *IEEE Conference on Computer Vision and Pattern Recognition*. 4954–4963.

[44] Soham Pal, Yash Gupta, Aditya Shukla, Aditya Kanade, Shirish Shevade, and Vinod Ganapathy. 2019. A framework for the extraction of deep neural networks by leveraging public data. *arXiv preprint arXiv:1905.09165* (2019).

[45] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. 2016. The limitations of deep learning in adversarial settings. In *IEEE European Symposium on Security and Privacy*. 372–387.

[46] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. "Why should I trust you?": Explaining the predictions of any classifier. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 1135–1144.

[47] Stuart Russell. 1998. Learning agents for uncertain environments. In *Annual Conference on Computational Learning Theory*. 101–103.

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

[49] Ruimin Shen, Yan Zheng, Jianye Hao, Zhaopeng Meng, Yingfeng Chen, Changjie Fan, and Yang Liu. 2020. Generating Behavior-Diverse Game AIs with Evolutionary Multi-Objective Deep Reinforcement Learning. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*. 3371–3377.

[50] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* 529, 7587 (2016), 484–489.

[51] Jianwen Sun, Tianwei Zhang, Xiaofei Xie, Lei Ma, Yan Zheng, Kangjie Chen, and Yang Liu. 2020. Stealthy and efficient adversarial attacks against deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 5883–5891.

[52] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2013. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199* (2013).

[53] Florian Tramèr, Fan Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. 2016. Stealing machine learning models via prediction apis. In *USENIX Security Symposium*. 601–618.

[54] Yusuke Uchida, Yuki Nagai, Shigeyuki Sakazawa, and Shin'ichi Satoh. 2017. Embedding watermarks into deep neural networks. In *International Conference on Multimedia Retrieval*. 269–277.

[55] Binghui Wang and Neil Zhenqiang Gong. 2018. Stealing hyperparameters in machine learning. In *IEEE Symposium on Security and Privacy*. 36–52.

[56] Ziyu Wang, Victor Bapst, Nicolas Heess, Volodymyr Mnih, Remi Munos, Koray Kavukcuoglu, and Nando de Freitas. 2016. Sample efficient actor-critic with experience replay. *arXiv preprint arXiv:1611.01224* (2016).

[57] Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning* 8, 3-4 (1992), 229–256.

[58] Yuhuai Wu, Elman Mansimov, Roger B Grosse, Shun Liao, and Jimmy Ba. 2017. Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation. In *Advances in Neural Information Processing Systems*. 5279–5288.

[59] Chaowei Xiao, Xinlei Pan, Warren He, Jian Peng, Mingjie Sun, Jinfeng Yi, Mingyan Liu, Bo Li, and Dawn Song. 2019. Characterizing attacks on deep reinforcement learning. *arXiv preprint arXiv:1907.09470* (2019).

[60] Cihang Xie, Zhishuai Zhang, Yuyin Zhou, Song Bai, Jianyu Wang, Zhou Ren, and Alan L Yuille. 2019. Improving transferability of adversarial examples with input diversity. In *IEEE Conference on Computer Vision and Pattern Recognition*. 2730–2739.

[61] Xiaofei Xie, Lei Ma, Felix Juefei-Xu, Minhui Xue, Hongxu Chen, Yang Liu, Jianjun Zhao, Bo Li, Jianxiong Yin, and Simon See. 2019. DeepHunter: a coverage-guided fuzz testing framework for deep neural networks. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*. 146–157.

[62] Honggang Yu, Kaichen Yang, Teng Zhang, Yun-Yun Tsai, Tsung-Yi Ho, and Yier Jin. 2020. CloudLeak: Large-scale deep learning models stealing through adversarial examples. In *Network and Distributed Systems Security Symposium*.

[63] Jialong Zhang, Zhongshu Gu, Jiyong Jang, Hui Wu, Marc Ph Stoecklin, Heqing Huang, and Ian Molloy. 2018. Protecting intellectual property of deep neural networks with watermarking. In *Asia Conference on Computer and Communications Security*. 159–172.

[64] Xiyue Zhang, Xiaofei Xie, Lei Ma, Xiaoning Du, Qiang Hu, Yang Liu, Jianjun Zhao, and Meng Sun. 2020. Towards characterizing adversarial defects of deep learning software from the lens of uncertainty. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*. IEEE, 739–751.

[65] Yan Zheng, Xiaofei Xie, Ting Su, Lei Ma, Jianye Hao, Zhaopeng Meng, Yang Liu, Ruimin Shen, Yingfeng Chen, and Changjie Fan. 2019. Wuji: Automatic online combat game testing using evolutionary deep reinforcement learning. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 772–784.

[66] Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, Joseph J Lim, Abhinav Gupta, Li Fei-Fei, and Ali Farhadi. 2017. Target-driven visual navigation in indoor scenes using deep reinforcement learning. In *IEEE International Conference on Robotics and Automation*. 3357–3364.