FLARE: Fingerprinting Deep Reinforcement Learning Agents using Universal Adversarial Masks

Buse G. A. Tekgul
Nokia Bell Labs & Aalto University
Espoo, Finland
buse.atli_tekgul@nokia-bell-labs.com

N. Asokan
University of Waterloo & Aalto University
Waterloo, Canada
asokan@acm.org

ABSTRACT

We propose FLARE, the first fingerprinting mechanism to verify whether a suspected Deep Reinforcement Learning (DRL) policy is an illegitimate copy of another (victim) policy. We first show that it is possible to find non-transferable, universal adversarial masks, i.e., perturbations, to generate adversarial examples that can successfully transfer from a victim policy to its modified versions but not to independently trained policies. FLARE employs these masks as fingerprints to verify the true ownership of stolen DRL policies by measuring an action agreement value over states perturbed by such masks. Our empirical evaluations show that FLARE is effective (100% action agreement on stolen copies) and does not falsely accuse independent policies (no false positives). FLARE is also robust to model modification attacks and cannot be easily evaded by more informed adversaries without negatively impacting agent performance. We also show that not all universal adversarial masks are suitable candidates for fingerprints due to the inherent characteristics of DRL policies. The spatio-temporal dynamics of DRL problems and sequential decision-making process make characterizing the decision boundary of DRL policies more difficult, as well as searching for universal masks that capture the geometry of it.

1 INTRODUCTION

Deep reinforcement learning (DRL) has emerged as a promising technique for building intelligent agents due to its ability to learn from and interact with high-dimensional input data. Following the work of Mnih et al. [26], which shows that DRL has exceeded human-level performance in Atari games, it has been successfully used in many real-world applications, including green data centers [17], autonomous driving [15] and robotic manipulation [14].

The commercial success and continuous improvement of DRL methods attract adversaries, leading them to look for and exploit vulnerabilities in DRL agents. DRL agents leverage the power of deep neural networks (DNNs) to improve agents' decision-making strategy, i.e., *policy*. Therefore, known vulnerabilities of DNNs might also be valid for agents' policies. For example, considerable research has been devoted to evasion attacks against DRL policies using adversarial examples [9, 12, 13, 30, 36], which are generally computed for input states that are representations of the environment received by the agent. Unlike evasion attacks and related defenses [18, 29, 44], few studies investigated the ownership of DRL models [3, 6, 21] against model piracy attacks. The high training costs of DRLs and their business advantage lead adversaries to steal models and redistribute them unauthorized ways. To deter adversaries, it is crucial to have the technical means to identify the true ownership of illegitimate copies of DRL agents.

Recently, DNN fingerprinting has been proposed as an ownership verification method [23, 31]. DNN fingerprinting aims to identify the inherent properties of the victim (original) model and use this information during verification. Current DNN fingerprinting methods leverage *conferrable adversarial examples* (CAE) [23] or *universal adversarial perturbations* (UAP fingerprinting) [31], as adversarial examples can characterize the DNN decision boundary. Conferrable adversarial examples [23] are a subclass of transferable adversarial examples that can successfully force the victim DNN model and its modified versions to make the same wrong predictions, but are not transferable to other independently trained models. Unlike CAE, UAP [31] obtains universal adversarial perturbations from both victim and suspected models, and produces a similarity score using contrastive learning. Both methods are shown to be effective and robust ownership verification approaches, but adapting them in DRL has challenges. First, both methods query suspected DNN models with adversarial version of input samples with different labels that are selected from the training set. However, this is not possible in DRL due to dynamic environments and continuous agent-environment interaction. Therefore, these methods require constructing a special verification setup that has unconventional environment dynamics and completely changes the trajectory of the agent regardless of the task. Second, there is no one-to-one mapping between the input states and the corresponding actions in DRL. This makes fingerprint generation challenging, since there is no single optimal action for any clean state and no desired incorrect action for any adversarial state either.

In this paper, we propose FLARE, *the first DRL fingerprinting scheme* designed for discrete reinforcement learning tasks and combines the idea behind CAE and UAP. The effectiveness of adversarial perturbations decreases as they transfer from one DRL policy or algorithm to another [12]. This implies that it would be possible to find adversarial examples that are not transferable across DRL agents. However, using individual non-transferable adversarial examples for ownership verification might be impractical due to the problems mentioned above. Therefore, FLARE aims to generate *non-transferable universal masks* as fingerprints, which are independent of input states, source actions, or target actions. Fingerprints computed by FLARE are instances of weaknesses that are inherent in the victim's DRL policy. FLARE leverages these weak points to verify the true ownership of *suspected* policies. During verification, suspected agents receive states modified by applying a universal mask from the victim's fingerprint in a small time window while trying to complete their task. FLARE verifies the true ownership if the similarity between the actions of the suspected agent and the victim agent in the same fingerprinted states is greater than a threshold value. FLARE does not change the training procedure,

and verification can be implemented at any time during deployment. Our main contributions are as follows:

(1) We propose FLARE, the first fingerprinting method to verify the ownership of DRL agents used in discrete tasks by leveraging non-transferable universal adversarial masks (Section 3). We show that FLARE is an effective ownership verification method with no false positives (Section 4.2).[1]

(2) We verify the robustness of FLARE against model modification attacks (e.g., fine-tuning and pruning) on 6 different DRL agents trained using two different games of the Arcade Learning Environment [4]. We also show that well-informed adversaries cannot easily evade verification without sacrificing agent performance, and FLARE is robust against false claims made by malicious accusers. (Section 4.3).

(3) We empirically demonstrate that universal adversarial perturbations generated by minimum-distance methods [27, 31] are not good candidates for DRL fingerprinting. These perturbations are not unique weaknesses of DRL policies by design and fail against model modification attacks (Section 5).

2 BACKGROUND

2.1 Deep Reinforcement Learning

2.1.1 Reinforcement Learning. A typical reinforcement learning (RL) problem is modeled as a 5-tuple Markov Decision Process (MDP) (S, A, P, R, γ), where S denotes the state space, A is the action space, P symbolizes the state transition probability (i.e., environment dynamics), R is the reward function, and $\gamma \in [0, 1]$ denotes the discount factor used to calculate the discounted cumulative reward, i.e., *return*. In this setting, the RL agent receives a state $s_t \in S$ at the time step t, performs an action $a_t \in A$, and then subsequently receives a reward r_{t+1} as well as the next state s_{t+1} based on $P(s_{t+1}|s_t, a_t)$. The objective of an RL agent is to maximize its expected return by interacting with the environment and to obtain an optimal policy $\pi(a|s) : S \rightarrow A$ that outputs an optimal action (the action that gives the maximum expected return over all actions) for any given state. During training, the policy is optimized recursively by calculating the expected return over states using the Bellman equation [34]. In this work, we consider states to be fully observable and finite-horizon tasks (i.e., an episode is completed when a stopping criterion is reached). Therefore, the discounted return at a time step t is calculated as $R_t = \sum_{k=t}^{T} \gamma^{k-t} r_k$ where T is the final time step in a single episode. We also focus on tasks with a discrete action space, where one-hot vectors can be used to distinguish one action from every other action.

2.1.2 Deep Reinforcement Learning (DRL). When the state space S is too complex and high-dimensional, deep neural networks (DNNs) can be useful to approximate policy $\pi(a|s)$. In this work, we assume that the environment is dynamic, as in real-world applications. Model-free DRL methods are the preferred approach in this setting, since these methods do not require estimating the dynamics of the environment. Two typical model-free DRL methods approximate π: value-based and policy-based methods. Value-based [26] methods approximate the action value function $Q^{\pi}(s, a)$ which computes

the estimated return of state s_t if the agent chooses the action a_t and then follows the current policy. The optimal policy is implicitly obtained once Q^{π} is optimized. Policy-based methods [25, 33] first parameterize the policy $\pi(a|s, \theta)$ and then optimize it by updating the parameters θ through the gradient ascent.

In this paper, we use π to symbolize the optimal policy obtained during training and $\hat{\pi}$ to denote the optimal action a_t decided by π for the input state s_t, where $a_t = \hat{\pi}(s_t)$.

2.2 Adversarial Examples

2.2.1 Adversarial Examples in DNN. An adversarial example x' is an intentionally modified input sample $x \in X$ with an imperceptible amount of noise r to force a DNN model $f : X \rightarrow Y$ into producing incorrect predictions $\hat{f}$. Targeted adversarial examples are labeled with y', the intended (incorrect) prediction, in advance to satisfy $y' = \hat{f}(x')$ and $y' \neq \hat{f}(x)$, while untargeted adversarial examples aim to evade the correct prediction, i.e., $\hat{f}(x') \neq \hat{f}(x)$. Untargeted adversarial examples against a victim DNN model f are computed by solving an optimization problem,

$$\operatorname*{argmax}_{x'} \mathcal{L}(f(x'), \hat{f}(x)) \text{ s.t.: } \|x' - x\|_p = \|r\|_p \leq \epsilon, \quad (1)$$

where $\mathcal{L}$ denotes the prediction loss of f. This formulation is used by *maximum-confidence* adversarial example generation methods [8] that maximize $\mathcal{L}$ while constraining the amount of perturbation with ϵ. On the contrary, the *minimum-distance* methods aim to minimize the sufficient amount of perturbation that changes the prediction [27].

An adversarial example x' calculated against one model f and successfully misleads it can *transfer* across other models, i.e., fools f^* that are trained for the same task. The transferability of an adversarial example increases when the source model f and the target models f^* learn similar decision boundaries [37]. Since maximum-confidence adversarial examples are misclassified with higher confidence, they have a higher transferability rate than minimum-confidence adversarial examples [8].

The definition of an adversarial example in DRL differs according to the target component of the victim agent and the overall goal [9, 12, 24, 30, 36, 40]. In this work, we consider adversarial states s' that mislead the policy π: $\hat{\pi}(s') \neq \hat{\pi}(s)$, $\|s' - s\|_p = \|r\|_p$, and set the norm p to ∞.

2.2.2 Universal Adversarial Perturbations. Instead of computing individual adversarial examples, Moosavi et al. [27] propose finding a perturbation vector r that fools the DNN model $\hat{f}(x + r) \neq \hat{f}(x)$ on almost all data points x sampled from the same distribution as the dataset $\mathcal{D}_{train}$ used for training f. The optimization problem in Equation 1 is modified to find universal perturbations as

$$\mathbb{P}_{x \sim \mu}(\hat{f}(x + r) \neq \hat{f}(x)) \geq \delta_r \text{ s.t.: } \|r\|_p \leq \epsilon, \quad (2)$$

where δ_r denotes the desired *fooling rate* of r for all x sampled from a dataset D with distribution μ.

Following Moosavi et al.'s initial work [27], several different techniques are proposed to generate universal adversarial perturbations. For example, Mopuri et al. [28] train a generative adversarial network to model the distribution of universal adversarial perturbations for a target DNN classification model and produce diverse

---

perturbations that achieve a high δ_r. Liu et al. [19] generate a universal adversarial perturbation that does not require training data and exploits the uncertainty of the model at each DNN layer.

2.3 Ownership Verification via Fingerprinting

Ownership verification in machine learning (ML) refers to a type of defense against model theft and extraction attacks by deterrence. Model owners can reduce the incentive for such attacks by identifying and verifying the true ownership of stolen models. DNN model fingerprinting is a well-known ownership verification technique. DNN fingerprinting methods identify unique knowledge that characterizes the victim model (fingerprint generation) and later use this information to verify whether the suspected model is derived from the victim model (fingerprint verification). For example, Ciao et al. [5] use adversarial example generation methods to extract data points near the decision boundary of DNN classifiers, label them as fingerprints, and utilize them along with their labels to detect piracy models. Lukas et al. [23] fingerprint DNN models through conferrable adversarial examples (CAE) that can successfully transfer from the source model to its modified versions, but not to other DNN models independently trained for the same classification task. To verify the fingerprint in a suspected model, CAE measures the error rate between the predictions of victim and suspected models, and the verdict is delivered based on a decision threshold. CAEs employ predictions of different independently trained models and modified versions of the victim model to compute fingerprints. Therefore, CAE has a high computational cost, since it requires training multiple modified and independent models to extract conferrable adversarial examples. Peng et al. [31] propose using universal adversarial perturbations (UAP) as fingerprints. During verification, previously computed UAPs for both victim and suspected models are mapped to a joint representation space, and contrastive learning is used to measure a similarity score in this projected space.

Adopting both UAP and CAE in DRL settings faces similar challenges. First, the verification episodes should include adversarial states that are completely different from each other, and also from a normal test episode during deployment. Second, in CAE, the predictions of multiple models having a good performance might be close to each other for the same input samples, since these models are trained over the same labeled dataset. However, there is no single predefined optimal action for input states in DRL. When agents receive the same state, they might act differently to perform the task due to their unique and different policies. Third, UAP fingerprinting uniformly selects data samples that are from different source classes and moves them towards different target classes in DNNs, but there is no one-to-one mapping between input states and corresponding optimal actions to obtain useful fingerprints in DRL settings.

3 METHODOLOGY

3.1 Adversary Model

The adversary $\mathcal{A}$'s goal is to obtain an illegal copy of the victim agent's ($\mathcal{V}$) policy $\pi_\mathcal{V}$ without being detected. $\mathcal{A}$ has economic incentives and aims to illegally monetize stolen policy $\pi_\mathcal{A}$

using a surrogate DRL agent. $\pi_\mathcal{V}$ can be leaked by exploiting hardware/software vulnerabilities [41] of different components within $\mathcal{V}$. Furthermore, $\mathcal{A}$ seeks to prevent traceback. Therefore, $\mathcal{A}$ attempts to degrade the effectiveness of possible ownership verification methods by modifying $\pi_\mathcal{A}$, without incurring any substantial drop in $\pi_\mathcal{A}$'s return.

3.1.1 Adversary's capabilities. $\mathcal{A}$ has computational capabilities and access to the similar environment that $\pi_\mathcal{V}$ was trained on, but it cannot reproduce the same training episodes. One can argue that $\mathcal{A}$ can also train its own policy, but we assume that it cannot obtain a policy as good as $\pi_\mathcal{V}$ due to nondeterminism (e.g., network architecture, difference in environment dynamics, DRL algorithm, hyperparameter selection, difference in computational resources, etc.). $\mathcal{A}$ presumes that there might be an ownership verification mechanism, but does not know the exact algorithm. Based on this assumption, we also consider the existence of *well-informed* adversaries[2] knowing that ownership verification is performed by fingerprinting and adversarial examples. If well-informed $\mathcal{A}$ knows the complete procedure of the fingerprinting process, then it can forge its own fingerprints to create ambiguity in verification. However, this could be prevented with FLARE if $\pi_\mathcal{V}$ and the corresponding fingerprints are securely time-stamped and registered in a bulletin or provided to a trusted third party [35].

3.1.2 Verifier's Capabilities. A verifier (judge, $\mathcal{J}$) is a trusted third party independent of both $\mathcal{V}$ and $\mathcal{A}$. Given a suspected DRL agent $\mathcal{S}$ with policy $\pi_\mathcal{S}$ and fingerprints provided by $\mathcal{V}$, the duty of $\mathcal{J}$ is to determine whether $\pi_\mathcal{S}$ can be traced back to $\pi_\mathcal{V}$ and demonstrate the true ownership. $\mathcal{J}$ has black-box access to $\mathcal{S}$, i.e., it does not know the algorithm and parameters of $\pi_\mathcal{S}$. $\mathcal{J}$ can modify the environment without introducing any temporal latency or suspending the task. If the verification uses time stamps, it provides anteriority to $\mathcal{J}$ to resolve any ambiguity. We also give $\mathcal{J}$ computational capabilities to train and search for independent policies used for the same task if there is a need to validate that fingerprints are unique to the original model and do not transfer to independent models. We also define that a good fingerprinting mechanism should satisfy the following requirements:

(1) **Effectiveness**: Successful ownership verification of stolen policies, i.e., maximizing true positives.
(2) **Integrity**: Avoiding accidental accusations of independently trained policies, i.e., minimizing false positives.
(3) **Robustness**: Withstanding model modification and evasion attacks. This is achieved if either the ownership of the modified policy is still successfully verified or the modification results in a substantial decrease in utility measured by the agent performance.

Fingerprinting algorithms do not necessarily aim for *utility* (i.e., maintaining the quality of the suspected model on fingerprints), as they typically use adversarial examples during verification [23, 31] and the desired outcomes for fingerprints contain incorrect

---

[2]ML literature commonly uses the term "adaptive" to refer to adversaries who are aware of deployed defenses. In security literature, it is customary to assume that *all* adversaries are aware of the defenses, and the term "adaptive" is used for adversaries who are able to *dynamically modify* their attack strategy based on what they learn about the defenses *during* the attack. We use the term "well-informed" to refer to such adversaries so that our usage does not conflict with either ML or security literature.

predictions. Therefore, we did not include utility as a requirement. However, we still restrict FLARE based on the utility concept, so that agents can still maintain their overall performance and complete the task without a significant performance degradation in episodes that include the verification phase.

3.2 Universal Adversarial Masks as Fingerprints

FLARE aims to find a set of adversarial masks that can fool the original agent in any input state to which it is added, but cannot transfer to independently trained agents. Lukas et al. [23] define a similar property for classifiers called "conferrability". Conferrable adversarial examples can transfer from the original classifier to its derivatives but not to independently trained classifiers. In contrast, FLARE does not generate individual adversarial examples but instead searches for universal adversarial masks that can be used to generate conferrable adversarial examples.

3.2.1 Fingerprint Generation. During fingerprint generation, FLARE first computes the universal adversarial mask using the original policy $\pi_\mathcal{V}$ and independently trained models $\pi_i, (i \in \mathcal{I})$ that have the same DNN architecture. FLARE aims to find a universal mask r that maximizes the loss function in Equation 3 and is bounded by ϵ in l_∞-norm.

$$\mathcal{L}(\pi_\mathcal{V}(s_t+r), \hat{\pi}_\mathcal{V}(s_t)) - \mathbb{1}_{(\hat{\pi}_\mathcal{V}(s_t)=\hat{\pi}_i(s_t))} \mathcal{L}(\pi_i(s_t+r), \hat{\pi}_i(s_t)) \quad (3)$$

The first part of Equation 3 maximizes the categorical cross-entropy loss between $\pi_\mathcal{V}$'s predictions for clean and adversarial states using the log-probability vector for all actions $\pi_\mathcal{V}$ in adversarial state and the performed action $\hat{\pi}_\mathcal{V}$ in the clean version of that state. The second part minimizes the categorical cross-entropy loss between adversarial states $s_t + r$ computed for π_i and their clean counterparts s_t only if the predicted action for s_t is the same for both $\pi_\mathcal{V}$ and π_i. The modified loss function ensures that the same $s_t + r$ cannot mislead $\pi_\mathcal{V}$ and π_i in the same way, even if $\hat{\pi}_i$ produces a suboptimal action. FLARE uses untargeted adversarial examples as fingerprints (see Section 2.2), so the solution of Equation 3 forces $\pi_\mathcal{V}$ into the incorrect action in $s_t + r$, but has a minimum effect on π_i. Multiple independently trained policies are used to calculate the second part of Equation 3 for each $i \in \mathcal{I}$ by taking the average of individual losses. A universal adversarial mask should also achieve a high fooling rate δ_r as presented in Equation 2.

To ensure universality, FLARE uses an approach similar to [30] when solving Equation 3. First, $\mathcal{V}$ completes one episode and the observed states are saved in a training set $\mathcal{D}_{flare}$. Then FLARE computes the average gradient of the loss function in Equation 3 w.r.t. k states randomly sampled from $\mathcal{D}_{flare}$. This enables FLARE to generate $\binom{len(\mathcal{D}_{flare})}{k}$ different universal adversarial masks as a fingerprint candidate. After generating the fingerprint candidate, FLARE checks its *non-transferability score*. We compute the non-transferability score (*nts*) for a universal adversarial mask r on an episode *eps* (that $\pi_\mathcal{V}$ follows) as

$$nts(r, eps) = \delta_{r,eps} \times max_{i \in \mathcal{I}}(1 - AA(\pi_\mathcal{V}, \pi_i, s, r)), \quad (4)$$

Algorithm 1 Fingerprint generation

Input: $\mathcal{D}_{flare}$: Fingerprint generation set
Output: FRL: Fingerprint list
1: parameters: $\tau_{nts}, \tau_\delta, n_{\text{episodes}}, n_{\text{FRL}}$
2: FRL = [ ].
3: **for** $eps \leq n_{\text{episodes}}$ **do**
4:      Generate $r_{candidate}$ from $\mathcal{D}_{flare}$
5:      Compute $nts(r_{candidate})$ using $\mathcal{I}$ and $\forall s_t \in eps$
6:      **if** $nts \geq \tau_{nts}$ **and** $\delta_{r_{candidate}} \geq \tau_\delta$ **then**
7:          Add $r_{candidate}$ into FRL
8:      **end if**
9:      **if** $len(\text{FRL}) == n_{\text{FRL}}$ **then**
10:          return FRL
11:      **end if**
12: **end for**
13: **return** FRL

---

where $\delta_{r,eps}$ refers to the fooling rate measured for $\pi_\mathcal{V}$ using all s observed in *eps*. *AA* denotes *action agreement* and is calculated as

$$AA(\pi_i, \pi_j, s, r) = \frac{1}{N} \sum_{t=0}^{t=N} \mathbb{1}_{(\hat{\pi}_i(s_t+r)=\hat{\pi}_j(s_t+r))}, \quad (5)$$

where N refers to the length of one full episode *eps* that $\pi_\mathcal{V}$ follows.

FLARE only accepts the candidate $r_{candidate}$ as a valid fingerprint if $nts(r_{candidate})$ is greater than a threshold value τ_{nts} and achieves a fooling rate $\delta_{r_{candidate}}$ higher than τ_δ over a single *eps*. How FLARE decides whether to include a universal adversarial mask in a fingerprint list FLR is presented in Algorithm 1.

3.2.2 Fingerprint Verification. For fingerprint verification, the verifier $\mathcal{J}$ has given a fingerprint set FLR. $\mathcal{J}$ first observes the interactions between the suspected agent $\mathcal{S}$ and the environment to estimate the total number of states N that occur during a single episode. Then, for each subsequent episode, $\mathcal{J}$ adds one fingerprint starting from a random state at time t_{start} over a short time window of length M to preserve the return in an acceptable range. $\mathcal{V}$ is also queried with the adversarial states $s_t + r$ that the suspected agent receives. For each fingerprint, *AA* is calculated as $1/M \sum_{t=t_{start}}^{t_{start}+M-1} AA(\pi_\mathcal{V}, \pi_\mathcal{S}, s_t, r)$. If *AA* for a single fingerprint exceeds a decision threshold $AA \geq 0.5$, that fingerprint produces supporting evidence to verify that the suspected model is the stolen copy. The final verdict (stolen vs. independent) is made based on the *majority vote*. FLARE also returns *AA* averaged on all fingerprints to quantify the confidence in the final decision. The verification procedure is summarized by Algorithm 2.

4 EMPIRICAL ANALYSIS

4.1 Experimental Setup

We evaluated FLARE using the Arcade Learning Environment (ALE) [4]. We selected two different games, Pong and MsPacman, from ALE to train agents with three different model-free DRL algorithms: A2C [25], DQN [26], and PPO [33]. Pong is a two-player game in which agents are trained to win against the computer, while MsPacman is a single-player game with the goal of achieving

Algorithm 2 Fingerprint verification

Input: FRL, $\pi_\mathcal{V}$, $\pi_\mathcal{S}$: Fingerprint list, victim and suspected policies
Output: AA, $Mvote$: action agreement, majority vote
1: $AA = [\;], Mvote = 0, Tvote = 0.$
2: Run a single episode with $\pi_\mathcal{S}$, save total number of states N
3: **for** i, r_i in $(range(\text{FRL}), \text{FRL})$ **do**
4:    $AA_i = 0.0$
5:    Generate random $t_{start} \in [0, min(N, N - M)]$
6:    Run a test episode with $\pi_\mathcal{S}$
7:    **while** test episode of $\pi_\mathcal{S}$ not finished **do**
8:       Calculate AA_i over time steps $t \in [t_{start}, t_{start} + M)$
9:    **end while**
10:    $Mvote += 1$ **if** $(AA_i \geq 0.5), Tvote += 1$
11:    Add AA_i into AA
12:    Decision: Stolen **if** $Mvote > (Tvote - Mvote)$
13: **end for**
14: **return** Decision, $Mvote$, mean and std of AA

the highest score without crashing into enemies. When constructing the state information, we applied the pre-processing methods proposed in [26]. Furthermore, for each victim $\mathcal{V}$, we independently trained five additional policies π_i ($i \in \mathcal{I}$) that have the same DNN architecture and the DRL algorithm as $\mathcal{V}$, and used them during fingerprint generation. In total, we obtained six victim policies and thirty independent policies. In Pong, the victim and the independent policies win the game with the highest score (+21). In MsPacman, it was harder to achieve similar high scores since states are more complex than Pong and depend on the position of multiple enemies. In both games, the score is used to quantify the agent's return. Appendix A.1 presents software/hardware requirements for reproduction, as well as the average performance of all agents.

During fingerprint generation in DQN, FLARE uses the DNN approximating Q value function. For other algorithms, FLARE selects the policy network (e.g., the actor network in A2C) to compute fingerprints. We set the maximum number of fingerprints $len(FRL)$ at 10, and the window size M at 40. The discussion on the choice of $len(FRL)$ and M is included in Appendix A.2. Other hyperparameters used in fingerprint generation are also listed in Appendix A.2. In our experimental setup, we used different random initialization for episodes used in training, fingerprint generation, verification, estimation of agent performance, modification attacks, and evasion attacks to ensure randomness in dynamic (and uncontrollable) environments.

4.2 Effectiveness and Integrity

Figure 1 summarizes various FLARE metrics (fooling rate δ, non-transferability score nts and action agreement AA calculated on different policies) for three different DRL algorithms. AA_{orig} denotes AA of the adversary's policy $\pi_\mathcal{A}$ which is identical to the victim policy $\pi_\mathcal{V}$. AA_{ind} (verification) refers to AA values of independent policies π_i that share the same DRL algorithm as $\pi_\mathcal{V}$ and are used in Algorithm 1 (5 policies for each $\mathcal{V}$). We use the remaining 10 independent policies (having a different DRL algorithm from $\mathcal{V}$) trained for the same task to calculate AA_{others} (verification). AA_{orig} is much higher than the threshold value 0.5, almost equal to

Figure 1: Various FLARE metrics averaged over 10 runs for all generated fingerprints. FLARE can successfully distinguish between the original model AA_{orig} and independent models AA_{ind}, AA_{others}, while achieving high fooling rate δ and non-transferability score nts.

1.0 in most cases. Furthermore, the average fooling rate of fingerprints is high, which proves that fingerprints successfully mislead $\pi_\mathcal{A}$. The average of AA_{ind} (verification) and AA_{others} (verification) are lower than 0.5 in all cases, and the majority vote is always "'not stolen (independent)" for any other π_i that is not $\pi_\mathcal{A}$. Results show that FLARE achieves a high detection rate while avoiding false accusations of independently trained ones. Thus, we conclude that FLARE satisfies the effectiveness and integrity requirements.

As shown in Figure 1, AA_{ind} and AA_{others} show different variances for three DRL algorithms. We found that one or two fingerprints seldom produce $AA \geq 0.5$ for $\pi_\mathcal{V}$ and π_i, although they behave differently in the same clean states. This reveals that a single fingerprint rarely represents the same weakness of two separate policies, and the number of fingerprints should be high enough to satisfy integrity considering this phenomenon.

During verification, we set the threshold value to 0.5 (a single fingerprint votes for "stolen" if $AA \geq 0.5$) over all experiments. However, it might be better to look at the full profile of the receiver operational characteristic (ROC) curves, which give a complete picture of the trade-off between false positive and true positive rates by varying the threshold value. We provide ROC curves for both Pong and MsPacman games in Appendix A.3.

Table 1: Average impact, *AA* and voting results (✓:Stolen, ✗: Independent) for piracy policies that are 1) fine-tuned over a different number of episodes and 2) pruned and then fine-tuned over 200 episodes. *AA* is averaged over 10 verification episodes, while impact is averaged over 10 test episodes. (🟩 : Successful verification with $AA \geq 0.75$, 🟦 : Successful verification with $0.75 \geq AA \geq 0.50$, 🟨 : Failed verification with high impact ≥ 0.4, 🟥 : Failed verification with low impact < 0.4)

Game, DRL method	Stats	Fine-tuning, # of episodes			Pruning and fine-tuning, pruning levels (%)			
		50	100	200	25	50	75	90
Pong, A2C	Impact	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	1.0 ± 0.0
	AA	0.95 ± 0.14	0.95 ± 0.14	0.94 ± 0.10	0.94 ± 0.14	0.91 ± 0.25	0.67 ± 0.42	0.28 ± 0.42
	Votes	10 ✓/ 0 ✗	10 ✓/ 0 ✗	10 ✓/ 0 ✗	10 ✓/ 0 ✗	9 ✓/ 1 ✗	6 ✓/ 4 ✗	3 ✓/ 7 ✗
Pong, DQN	Impact	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.8 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
	AA	0.94 ± 0.05	0.89 ± 0.14	0.90 ± 0.17	0.88 ± 0.16	0.66 ± 0.38	0.09 ± 0.17	0.27 ± 0.4
	Votes	10 ✓/ 0 ✗	10 ✓/ 0 ✗	9 ✓/ 1 ✗	10 ✓/ 0 ✗	7 ✓/ 3 ✗	1 ✓/ 9 ✗	3 ✓/ 7 ✗
Pong, PPO	Impact	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
	AA	0.88 ± 0.23	0.89 ± 0.25	0.88 ± 0.30	0.78 ± 0.35	0.67 ± 0.35	0.65 ± 0.41	0.71 ± 0.39
	Votes	9 ✓/ 1 ✗	9 ✓/ 1 ✗	9 ✓/ 1 ✗	7 ✓/ 3 ✗	7 ✓/ 3 ✗	6 ✓/ 4 ✗	7 ✓/ 3 ✗
MsPacman, A2C	Impact	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.39 ± 0.19	0.03 ± 0.10	0.30 ± 0.15	0.73 ± 0.11
	AA	0.82 ± 0.16	0.75 ± 0.29	0.62 ± 0.35	0.71 ± 0.28	0.65 ± 0.39	0.72 ± 0.26	0.59 ± 0.23
	Votes	9 ✓/ 1 ✗	8 ✓/ 2 ✗	6 ✓/ 4 ✗	6 ✓/ 4 ✗	7 ✓/ 3 ✗	8 ✓/ 2 ✗	6 ✓/ 4 ✗
MsPacman, DQN	Impact	0.79 ± 0.11	0.83 ± 0.02	0.87 ± 0.03	0.79 ± 0.11	0.74 ± 0.09	0.86 ± 0.01	0.71 ± 0.43
	AA	0.23 ± 0.34	0.15 ± 0.28	0.16 ± 0.31	0.38 ± 0.44	0.00 ± 0.01	0.59 ± 0.46	0.42 ± 0.42
	Votes	2 ✓/ 8 ✗	1 ✓/ 9 ✗	2 ✓/ 8 ✗	4 ✓/ 6 ✗	0 ✓/ 10 ✗	6 ✓/ 4 ✗	4 ✓/ 6 ✗
MsPacman, PPO	Impact	0.85 ± 0.11	0.40 ± 0.26	0.51 ± 0.08	0.52 ± 0.15	0.57 ± 0.04	0.62 ± 0.05	0.66 ± 0.19
	AA	0.43 ± 0.36	0.11 ± 0.16	0.25 ± 0.32	0.26 ± 0.36	0.33 ± 0.38	0.31 ± 0.32	0.13 ± 0.20
	Votes	4 ✓/ 6 ✗	0 ✓/ 10 ✗	3 ✓/ 7 ✗	3 ✓/ 7 ✗	3 ✓/ 7 ✗	4 ✓/ 6 ✗	1 ✓/ 9 ✗

Utility : As stated in Section 3.1, we do not consider utility a requirement for FLARE. However, based on the definition in [16], we measure the *impact* of the verification on the victim agent to ensure that it does not fail the task during verification. We measure the impact as:

$$Impact = \frac{Return_{\mathcal{V}(test)} - Return_{\mathcal{V}(verification)}}{Return_{\mathcal{V}(test)} - Return_{\mathcal{V}_{min}(test)}}. \quad (6)$$

$Return_{\mathcal{V}(test)}$ and $Return_{\mathcal{V}(verification)}$ are the average return of $\mathcal{V}$ in an episode initialized with the same start state (and with the same environment dynamics) with or without the verification. $Return_{\mathcal{V}_{min}(test)}$ is the return of $\mathcal{V}$ if it chooses the worst possible actions for each state in the same episode. The results presented in Appendix A.4 show that the average impact on agent performance is 0.02 and 0.22 in MsPacman and Pong, respectively. We also found that the return never drops to $Return_{\mathcal{V}_{min}(test)}$ during verification. Thus, we conclude that agents continue their task without a significant impact after the verification phase ends.

4.3 Robustness

4.3.1 Robustness Against Model Modification Attacks. Adversary $\mathcal{A}$ could modify the stolen policy $\pi_{\mathcal{A}}$ by carefully retraining it to preserve agent performance while trying to suppress evidence used for verification. We consider two common types of model modification attacks, fine-tuning [32] and weight pruning [11], where $\mathcal{A}$ is aware of the existence of an ownership verification technique but does not know the type of it. We implemented fine-tuning by retraining $\pi_{\mathcal{A}}$ in an additional 200 episodes and decreasing the learning rate by 100 to maintain agent performance. For pruning, we first performed global pruning, i.e., removed a percentage of the lowest connections across the DNN model. After pruning, we fine-tuned the pruned model over 200 episodes.

To evaluate the robustness requirement, we computed the majority vote and *AA* values of the stolen $\pi_{\mathcal{A}}$ and modified policies $\pi_{\mathcal{A}^*}$ on the verification episodes. We also measured the impact of the modification on model utility (agent performance) by changing Equation 6 to:

$$Impact = \frac{Return_{\mathcal{A}(test)} - Return_{\mathcal{A}^*(test)}}{Return_{\mathcal{A}(test)} - Return_{A_{min}(test)}}, \quad (7)$$

where $Return_{\mathcal{A}^*}$ is the return of stolen and modified policy, and $Return_{\mathcal{A}}$ denotes the return of stolen (unmodified) policy over the same test episodes. Based on the results on the impact of verification on utility, we generously set the maximum allowable impact as 0.4 for modification attacks, indicating that $Return_{\mathcal{A}^*(test)}$

Figure 2: *AA* **and return when attacker implements random action with different ratios, Visual Foresight (VF), and VF+suboptimal action as evasion against FLARE.** *AA* **is averaged in 10 verification episodes, whereas the return is averaged in 10 test episodes. Solid lines represent the return while dashed lines refer to** *AA*. **Each plot includes three solid and dashed lines (some of which overlap), and different markings on these lines refer to a specific evasion method.**

would fall a little more than halfway between $Return_{\mathcal{A}(test)}$ and $Return_{\mathcal{A}_{min}(test)}$.

Table 1 shows the robustness evaluation of FLARE against model modification attacks. As shown in the table, FLARE successfully verifies fine-tuned Pong agents with high *AA* values. FLARE usually results in a failed verification of fine-tuned MsPacman agents. However, the impact of modification is exceptionally high for these cases. A similar conclusion can be drawn from the pruning results. An increase in the pruning level negatively affects the verification by decreasing its *AA* values. However, the impact of pruning is too high in three cases in Pong and most of the cases in MsPacman, despite failed verification. Based on our robustness definition in Section 3.1, we conclude that FLARE is robust against model modification attacks.

4.3.2 Robustness Against Evasion Attacks and Well-informed Adversaries. $\mathcal{A}$ can evade verification by discovering individual inputs used for verification or adapt the agent's behavior to avoid a successful verification. For evasion, $\mathcal{A}$ should have more information about the ownership verification procedure. In our setup, $\mathcal{A}$ knows that the ownership verification is done via FLARE but is unaware of the exact adversarial mask used during verification. Based on this information, the simplest evasion attack is performing suboptimal actions with a pre-defined random action ratio on each episode. Figure 2 confirms that the increase in the random action ratio causes a decrease in agent performance (lower return) despite successful

evasion. Therefore, FLARE is robust against evasion via suboptimal action return.

Evasion attacks can combine detecting adversarial examples (i.e., fingerprints used for verification) and then performing either suboptimal actions or restoring original actions. We employ Visual Foresight (VF) [18] to carry out this attack. VF predicts the next states and the associated probability distribution of actions by looking at a history of previous states and observed actions. If the distance between the predicted and current action distribution is large, VF detects that state as adversarial, and performs the predicted action instead of the current one as a recovery mechanism. Figure 2 shows that the use of VF does not affect the agent performance. However, the high values of *AA* shown in the figure justifies that VF cannot recover agent performance when states are perturbed with non-transferable, universal adversarial masks and fail to evade verification. This is because the collected history of previous states consists of adversarial inputs, which might lead to the original (incorrect) action even if the adversarial state is detected correctly [36]. For this reason, we also evaluated the case where VF chooses a suboptimal action (VF + suboptimal action) instead of the one predicted during recovery. Figure 2 shows that it decreases *AA* more than VF, but *AA* is not too low to evade verification and change the final verdict.

Finally, we evaluated FLARE against the most well-informed adversaries that can improve the robustness of the policy against l_∞

norm adversarial perturbations by adversarial training. For adversarial training, we implemented one of the recent state-of-the-art methods, RADIAL-RL [29]. We choose to implement RADIAL-RL for DQN agents, because these are shared cases between our and the authors' experiments. RADIAL-DQN (RADIAL-RL designed for DQN) first obtains a policy without adversarial training and then fine-tunes the policy by incorporating an adversarial loss term into the loss function that is minimized during training. In our setting, $\mathcal{A}$ performs RADIAL-DQN by skipping the first step and fine-tunes the stolen policy $\pi_{\mathcal{A}}$ using adversarial loss. We adopted the open source repository of the authors [3] in our framework, did not change the hyperparameters used in RADIAL-DQN, and saved both agents with the best performance and the final agent after RADIAL-DQN was completed.

The first two rows of Table 2 summarize the impact, AA values, and the votes for agents modified through RADIAL-DQN. The results indicate that $\mathcal{A}$ can evade verification by making $\pi_{\mathcal{A}}$ more robust to adversarial states in Pong. $\mathcal{A}$ obtains an improved policy for MsPacman (3rd column, negative impact: higher reward), but cannot evade verification. This outcome is not surprising, as DNN fingerprinting has limitations against adaptive adversaries that perform adversarial training [23]. Then, we considered an alternative scenario where $\mathcal{V}$ fine-tunes its policy with RADIAL-DQN, saves the best agent, and generates fingerprints for this agent (RDQN). The last two rows of Table 2 show the verification results when $\mathcal{A}$ implements RADIAL-DQN against adversarially robust victim agents. In this case, $\mathcal{A}$ cannot evade verification without affecting the agent's performance. Therefore, although FLARE is limited against adversarial training, it satisfies the robustness requirement when fingerprinting adversarially robust victim agents.

4.3.3 Robustness Against False Claims. Liu et al. [20] show that malicious accusers can produce fake fingerprints that can pass the ownership verification test against independent models in many ownership verification schemes, including CAE [23]. Therefore, we also evaluated the robustness of FLARE against malicious accusers by generating fingerprints for the accuser's policy without maximizing the loss for independent policies (Equation 3) and not measuring the non-transferability score (Algorithm 1, line 6), which is similar to the setup proposed in [20] to evaluate CAE. We selected one of the five independent policies $\pi_i, i \in \mathcal{I}$ that behaves the closest to $\pi_{\mathcal{V}}$ in the test episodes and has the same DRL algorithm as the accuser policy and other independent policies as $\mathcal{J}$'s control set. As shown in Table 3, the malicious accuser cannot falsely claim ownership of $\mathcal{V}$ for the perturbation constraint set in FLARE ($\epsilon = 0.05$), except the PPO agent trained for Pong. If the perturbation constraint becomes larger ($\epsilon \geq 0.1$), then the accuser's false fingerprints transfer to other models in those cases. Having $\mathcal{J}$ perform an additional check that the size of the adversarial mask does not exceed a prescribed bound can mitigate against false claims attacks for FLARE. Table 3 also indicates that adversarial states have a higher transferability rate between PPO algorithms compared to others. In these cases, $\mathcal{J}$ can train or search for other independent PPO policies for the same task as suggested in [20], and it can reject the claim if the accuser's fingerprints falsely verify all independent models. Therefore, we conclude that FLARE is not

[3]https://github.com/tuomaso/radial_rl_v2

Table 2: Average impact, *AA* and voting results for stolen policies modified by RADIAL-DQN. Results are reported for both the agent with the best performance during RADIAL-DQN (3rd column) and the final agent obtained after RADIAL-DQN finishes (4th column). *AA* is averaged on 10 verification episodes and impact is averaged over 10 test episodes. (*: improved policy, ▇ : Successful verification with $AA \geq 0.75$, ▇ : Successful verification with $0.75 \geq AA \geq 0.50$, ▇ : Failed verification with high impact ≥ 0.4, ▇ : Failed verification with low impact < 0.4)

Game, DRL method	Stats	Best Agent	Final Agent
Pong, RADIAL-DQN	Impact	0.0 ± 0.0	0.0 ± 0.0
	AA	0.04 ± 0.06	0.04 ± 0.06
	Votes	0 ✓/ 10 ✗	0 ✓/ 10 ✗
MsPacman, RADIAL-DQN	Impact	$-0.16 \pm 0.03^*$	0.39 ± 0.03
	AA	0.59 ± 0.40	0.29 ± 0.31
	Votes	6 ✓/ 4 ✗	4 ✓/ 6 ✗
Pong, RADIAL-RDQN	Impact	0.0 ± 0.0	0.0 ± 0.0
	AA	0.84 ± 0.21	0.89 ± 0.17
	Votes	8 ✓/ 2 ✗	9 ✓/ 1 ✗
MsPacman, RADIAL-RDQN	Impact	0.15 ± 0.04	0.55 ± 0.06
	AA	0.61 ± 0.34	0.09 ± 0.18
	Votes	7 ✓/ 3 ✗	1 ✓/ 9 ✗

susceptible to false claims with a simple additional countermeasure on ϵ and non-transferability check based on the DRL algorithm.

Model extraction attacks in DRL : In this work, we limit the scope to the adversary model described in Section 3.1 and do not consider model extraction attacks against DRL policies through imitation learning [7]. Nevertheless, we tried to implement the model extraction attack proposed by Chen et al. [7], but were unable to obtain good stolen policies, which could be due to the simpler tasks chosen in the setup of the original work. Chen et al. [7] experimentally show that adversarial examples can successfully transfer from stolen policies to the victim policy if they share the same DRL algorithm. Their preliminary results provide insight into the possibility of preserving fingerprints during the extraction of the DRL model. Thus, we leave the construction of effective DRL model extraction attacks and evaluate the robustness of FLARE against these attacks for future work.

5 TRANSFERABILITY OF UNIVERSAL MASKS

The fingerprint generation process in FLARE is based on maximum-confidence adversarial example generation techniques and is similar to Fast Gradient Sign Method (FGSM) [12], since FLARE averages the gradient of Equation 3 w.r.t. randomly selected states. As presented in Section 2.2, maximum-confidence adversarial examples have a higher transferability rate than minimum-confidence examples. FGSM is a maximum-confidence method itself; however,

Table 3: *AA* values (averaged over 10 verification episodes) and voting results for false claims against victim $\mathcal{V}$, and independent $\mathcal{I}$ policies with different perturbation constraint ϵ values. (The cases where a false claim succeeds are shown as follows: ▮ : False claim with $AA \geq 0.75$, ▮ : False claim with $0.75 \geq AA \geq 0.50$)

Game, DRL method		ϵ vs. *AA* (Votes)			
		0.05	**0.1**	**0.2**	**0.5**
Pong,	$\mathcal{V}$	0.45 ± 0.47 (5 ✓/ 5 ✗)	0.49 ± 0.49 (5 ✓/ 5 ✗)	0.40 ± 0.49 (4 ✓/ 6 ✗)	0.40 ± 0.49 (4 ✓/ 6 ✗)
A2C	$\mathcal{I}$, avg.	0.32 ± 0.36 (3 ✓/ 7 ✗)	0.38 ± 0.45 (3 ✓/ 7 ✗)	0.30 ± 0.41 (3 ✓/ 7 ✗)	0.28 ± 0.43 (3 ✓/ 7 ✗)
Pong,	$\mathcal{V}$	0.37 ± 0.42 (4 ✓/ 6 ✗)	0.37 ± 0.45 (3 ✓/ 7 ✗)	0.33 ± 0.45 (3 ✓/ 7 ✗)	0.40 ± 0.49 (4 ✓/ 6 ✗)
DQN	$\mathcal{I}$, avg.	0.01 ± 0.18 (1 ✓/ 9 ✗)	0.07 ± 0.22 (1 ✓/ 9 ✗)	0.05 ± 0.19 (1 ✓/ 9 ✗)	0.05 ± 0.19 (1 ✓/ 9 ✗)
Pong,	$\mathcal{V}$	0.56 ± 0.39 (5 ✓/ 5 ✗)	0.68 ± 0.42 (7 ✓/ 3 ✗)	0.76 ± 0.38 (8 ✓/ 2 ✗)	0.78 ± 0.39 (8 ✓/ 2 ✗)
PPO	$\mathcal{I}$, avg.	0.56 ± 0.36 (6 ✓/ 4 ✗)	0.59 ± 0.38 (6 ✓/ 4 ✗)	0.59 ± 0.38 (6 ✓/ 4 ✗)	0.52 ± 0.41 (6 ✓/ 4 ✗)
MsPacman,	$\mathcal{V}$	0.00 ± 0.00 (0 ✓/10 ✗)	0.03 ± 0.05 (0 ✓/10 ✗)	0.14 ± 0.29 (1 ✓/9 ✗)	0.09 ± 0.22 (1 ✓/9 ✗)
A2C	$\mathcal{I}$, avg.	0.15 ± 0.56 (1 ✓/9 ✗)	0.14 ± 0.21 (1 ✓/9 ✗)	0.13 ± 0.30 (2 ✓/8 ✗)	0.21 ± 0.36 (2 ✓/8 ✗)
MsPacman,	$\mathcal{V}$	0.23 ± 0.36 (2 ✓/8 ✗)	0.0 ± 0.0 (0 ✓/10 ✗)	0.0 ± 0.0 (0 ✓/10 ✗)	0.0 ± 0.0 (0 ✓/10 ✗)
DQN	$\mathcal{I}$, avg.	0.26 ± 0.24 (2✓/8 ✗)	0.19 ± 0.26 (2✓/8 ✗)	0.15 ± 0.29 (1✓/9✗)	0.24 ± 0.26 (3✓/7✗)
MsPacman,	$\mathcal{V}$	0.19 ± 0.18 (1 ✓/ 9 ✗)	0.26 ± 0.31 (3 ✓/ 7 ✗)	0.38 ± 0.37 (4 ✓/ 6 ✗)	0.07 ± 0.21 (1 ✓/ 9 ✗)
PPO	$\mathcal{I}$, avg.	0.10 ± 0.11 (0 ✓/10 ✗)	0.50 ± 0.39 (5 ✓/5 ✗)	0.74 ± 0.40 (8 ✓/2 ✗)	0.80 ± 0.20 (8 ✓/2 ✗)

during the computation of universal, non-transferable adversarial masks, the effect of the high-sensitivity directions obtained from the most confident adversarial examples is diminished by others. Nevertheless, we analyzed whether minimum-confidence adversarial masks in DNNs can be useful for DRL fingerprinting. For that reason, we changed the universal mask generation r, (Algorithm 1, line 4) with Universal Adversarial Perturbation (UAP) [27] by implementing the method proposed for DRL settings [36].

We found that FLARE with UAP satisfies the effectiveness and integrity requirements for all agents, except the DQN agent trained for MsPacman. It was impossible to obtain an adversarial example with the perturbation constraint used in FLARE ($\epsilon = 0.05$) against this agent, but increasing it leads to transferable adversarial examples and false positives. The real issue with UAP emerges when the adversary $\mathcal{A}$ modifies the stolen policy $\pi_{\mathcal{A}}$ with model modification attacks. Due to its minimum-distance property, UAP finds the smallest high-sensitivity directions belonging to the closest incorrect class (or discrete actions in DRL), and generally the resulting r is smaller than ϵ. Therefore, a small change in $\pi_{\mathcal{A}}$ negatively affects the robustness of UAP. Contrary to UAP, FLARE shifts the source sample using the maximum amount of perturbation ϵ, forces $\pi_{\mathcal{A}}$ to perform the same incorrect action and is more robust against model modification attacks. We illustrate this problem in Figure 3.

One of the main reasons why FLARE has better robustness stems from the fact that the input space embeddings in DRL are not as separable as in DNN [2]. In DRL, although the input states are spatially similar, they often result in different actions. DRL agents optimize policies using both input state and environment dynamics and act upon spatio-temporal abstractions [43]. FLARE identifies discontinuities in the optimal policy and computes an adversarial state that is spatially similar to the source state but far from it in temporal dimension. UAP typically explores adversarial pockets

Figure 3: Depiction of fingerprints generated by UAP and FLARE for a DRL policy and three available actions. UAP moves the source sample in the direction of the closest incorrect action, and typically this movement is less than the perturbation constraint (denoted by circles). In contrast, FLARE shifts the source sample to the same action, which is irrelevant to the original action, while using the maximum value of the perturbation constraint.

that are closer in the spatial domain due to its minimum-distance strategy. Therefore, it cannot withstand model modification attacks that preserve the spatio-temporal abstractions and slightly change the sequential strategy.

We provide experimental results for our discussion in Table 4. This table compares the fooling rate and action agreement *AA* for adversarial states used in the verification of fine-tuned policies.

Table 4: Comparison of UAP and FLARE based on fooling rate (measured for the victim policy) and action agreement *AA*. Both the fooling rate and *AA* are averaged using adversarial states (fingerprints) in 10 verification episodes. The higher fooling rate and *AA* values are highlighted in green. Matched actions: Cases where victim and modified policies perform the same action for the same state. Different actions: Cases where victim and modified policies perform different actions for the same state.

Game, DRL Method	UAP				FLARE			
(Fine-tuned	Matched actions		Different actions		Matched actions		Different actions	
over 200 eps.)	Fooling rate	AA	Fooling rate	AA	Fooling rate	AA	Fooling rate	AA
Pong, A2C	0.79 ± 0.09	0.78 ± 0.08	0.89 ± 0.07	0.69 ± 0.13	0.95 ± 0.10	0.94 ± 0.09	0.85 ± 0.12	0.92 ± 0.16
Pong, DQN	0.69 ± 0.11	0.12 ± 0.10	0.55 ± 0.18	0.24 ± 0.14	0.89 ± 0.13	0.92 ± 0.18	0.93 ± 0.07	0.94 ± 0.09
Pong, PPO	0.86 ± 0.05	0.40 ± 0.21	0.82 ± 0.14	0.41 ± 0.13	0.91 ± 0.03	0.98 ± 0.08	0.90 ± 0.07	0.87 ± 0.3
MsPacman, A2C	0.76 ± 0.32	0.53 ± 0.42	0.91 ± 0.13	0.58 ± 0.42	0.68 ± 0.36	0.64 ± 0.36	0.64 ± 0.42	0.55 ± 0.43

We chose to report the results for fine-tuned policies from Table 1 considering the acceptable impact range (< 0.4) on the modified agent's performance. Matched actions refers to situations where both victim and modified policies perform the same action for the same input state without any added fingerprints. In contrast, different actions refer to cases where victim and modified policies behave differently for the same input state. Table 4 shows that the fooling rate of UAP is lower than FLARE in almost all cases. This supports our first claim regarding the robustness of UAP and FLARE. The columns labeled with *AA* show the action agreement where the fingerprint successfully misleads the victim policy. In this case, the ideal *AA* value for matched actions would be 1.0. As can be seen from Table 4, FLARE reaches much higher *AA* values for matched actions than UAP. Surprisingly, FLARE attains higher *AA* values for different actions as well. This shows that, even if the fine-tuned policy successfully changes the agent's behavior, the adversarial states generated by FLARE force the policy to perform the same incorrect action. The same conclusion cannot be drawn from the UAP results, as the *AA* values reported for the same actions are lower than FLARE even in the case with the higher fooling rate.

Based on this discussion, we conjecture that using minimum-distance adversarial examples to fingerprint DRL agents requires either adding modified policies to the loss function in Equation 3, or considering the temporal structure of the policy while finding the high-sensitivity directions. The latter option also opens a new space of adversarial examples that can exploit temporal abstractions learned by DRL policies.

6 RELATED WORK

Adversarial Examples in DRL : Recent work has shown that DRL policies are vulnerable to adversarial examples generated for agents' states [12, 36] or actions [40] in single-agent environments, or produce natural adversarial states by exploiting other agents in multi-agent settings [9]. Other studies focus on perturbing the dynamics of the environment by modifying the environment conditions [24, 30]. DRL adversarial training [29, 45] has been considered as a mitigation, but adversarially robust policies were found to be more vulnerable to high-sensitivity directions caused by a natural change in the environment [16].

Ownership Verification via Model Watermarking : Model watermarking has become a widely known ownership verification procedure for DNNs [1, 21, 45]. Model watermarking embeds traceable information (i.e. watermark) into the DNN by either directly inserting it

into model parameters or adding unique knowledge into a small subset of the training set. During ownership verification, the existence of the watermark is proven on illegitimate copies. Previous DRL ownership verification methods adapt model watermarking techniques. For example, Behzadan et al. [3] propose the embedding of sequential states that are separate from the main environment as watermarks during training. However, watermark verification also requires a different environment, and there is no guarantee that watermarks will be retained while learning complex tasks. Chen et al. [6] obtain a sequence of damage-free states as watermarks that are sampled from the same environment and do not impact agent performance. During verification, the authors compare the action probability distributions given by both the victim and the suspected agents over these sequential states. However, this watermarking method requires modifying both the training process and the reward function.

Although model watermarking is considered a practical solution to protect DNN ownership, many studies have shown [22, 42] that they cannot withstand well-informed adversaries and model modification attacks. Compared to watermarking, DNN fingerprinting methods show improved robustness to model modification and extraction attacks [23, 31]. Furthermore, fingerprinting does not change the training procedure unlike watermarking. However, there is no prior work applying fingerprinting as an ownership verification method in DRL.

Ownership Verification of Large models via Fingerprinting: Although FLARE is specifically designed for DRL, we conjecture that universal and non-transferable adversarial masks can be useful for fingerprinting, e.g., large language models. For example, Wallace et al. [38] show the availability of context-independent universal adversarial triggers that force large language models to produce incorrect results. Similarly, Gu et al. [10] demonstrate that universal adversarial patches can fool vision transformers. If universality is restricted with the non-transferability requirement, then the generated adversarial masks will profile the global behavior of large models and can be used in ownership verification.

7 CONCLUSION

In this paper, we propose FLARE, the first fingerprint method that can be used for ownership verification of DRL policies, and show the existence of non-transferable universal adversarial masks in DRL settings. We empirically demonstrate that our fingerprints are efficient and do not accidentally accuse independently trained

models. Adversarial training is the only method that evades verification by making policies robust to adversarial examples. However, our experiments show that the fingerprints obtained by FLARE for robust policies are persistent. We hypothesize that FLARE can be extended to continuous tasks, where the verifier can check how much the suspected agent deviates from the original action value, and we leave this for future work.

A promising direction for future work related to DRL fingerprinting is to study whether an intentional change in environment conditions can be useful candidates for fingerprints. DRL policies show decreased robustness when deployed in a different environment and include high-sensitivity directions due to natural causes. This vulnerability leads to model evasion attacks via natural adversarial examples, but can also be leveraged to learn natural (and non-transferable) fingerprints for ownership verification. We believe that our study can create more interest in securing DRL agents using novel ownership verification methods against possible model piracy and extraction attacks.

REFERENCES

[1] Yossi Adi, Carsten Baum, Moustapha Cissé, Benny Pinkas, and Joseph Keshet. 2018. Turning Your Weakness Into a Strength: Watermarking Deep Neural Networks by Backdooring. In *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018*, William Enck and Adrienne Porter Felt (Eds.). USENIX Association, 1615–1631.

[2] Raghuram Mandyam Annasamy and Katia Sycara. 2019. Towards Better Interpretability in Deep Q-Networks. *Proceedings of the AAAI Conference on Artificial Intelligence* 33, 01 (Jul. 2019), 4561–4569. https://ojs.aaai.org/index.php/AAAI/article/view/4377

[3] Vahid Behzadan and William H. Hsu. 2019. Sequential Triggers for Watermarking of Deep Reinforcement Learning Policies. *CoRR* abs/1906.01126 (2019). arXiv:1906.01126 http://arxiv.org/abs/1906.01126

[4] Marc G. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. 2013. The Arcade Learning Environment: An Evaluation Platform for General Agents. *J. Artif. Intell. Res.* 47 (2013), 253–279. https://doi.org/10.1613/jair.3912

[5] Xiaoyu Cao, Jinyuan Jia, and Neil Zhenqiang Gong. 2021. IPGuard: Protecting Intellectual Property of Deep Neural Networks via Fingerprinting the Classification Boundary. In *ASIA CCS '21: ACM Asia Conference on Computer and Communications Security, Virtual Event, Hong Kong, June 7-11, 2021*, Jiannong Cao, Man Ho Au, Zhiqiang Lin, and Moti Yung (Eds.). ACM, 14–25.

[6] Kangjie Chen, Shangwei Guo, Tianwei Zhang, Shuxin Li, and Yang Liu. 2021. Temporal Watermarks for Deep Reinforcement Learning Models. In *20th International Conference on Autonomous Agents and Multiagent Systems, Virtual Event, United Kingdom, May 3-7, 2021*, Frank Dignum, Alessio Lomuscio, Ulle Endriss, and Ann Nowé (Eds.). ACM, 314–322. https://doi.org/10.5555/3463952.3463994

[7] 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)*. Association for Computing Machinery, New York, NY, USA, 307–319.

[8] Ambra Demontis, Marco Melis, Maura Pintor, Matthew Jagielski, Battista Biggio, Alina Oprea, Cristina Nita-Rotaru, and Fabio Roli. 2019. Why Do Adversarial Attacks Transfer? Explaining Transferability of Evasion and Poisoning Attacks. In *28th USENIX Security Symposium, USENIX Security 2019, Santa Clara, CA, USA, August 14-16, 2019*, Nadia Heninger and Patrick Traynor (Eds.). USENIX Association, 321–338.

[9] Adam Gleave, Michael Dennis, Cody Wild, Neel Kant, Sergey Levine, and Stuart Russell. 2020. Adversarial Policies: Attacking Deep Reinforcement Learning. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net. https://openreview.net/forum?id=HJgEMpVFwB

[10] Jindong Gu, Volker Tresp, and Yao Qin. 2022. Are Vision Transformers Robust to Patch Perturbations?. In *Computer Vision – ECCV 2022*, Shai Avidan, Gabriel Brostow, Moustapha Cissé, Giovanni Maria Farinella, and Tal Hassner (Eds.). Springer Nature Switzerland, Cham, 404–421.

[11] Song Han, Jeff Pool, John Tran, and William J. Dally. 2015. Learning Both Weights and Connections for Efficient Neural Networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1* (Montreal, Canada) *(NIPS'15)*. MIT Press, Cambridge, MA, USA, 1135–1143.

[12] Sandy H. Huang, Nicolas Papernot, Ian J. Goodfellow, Yan Duan, and Pieter Abbeel. 2017. Adversarial Attacks on Neural Network Policies. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Workshop Track Proceedings*. OpenReview.net. https://openreview.net/forum?id=ryvlRyBKl

[13] Matthew Inkawhich, Yiran Chen, and Hai Li. 2020. Snooping Attacks on Deep Reinforcement Learning. In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems* (Auckland, New Zealand) *(AAMAS '20)*. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 557–565.

[14] Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, and Sergey Levine. 2018. Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation. In *2nd Annual Conference on Robot Learning, CoRL 2018, Zürich, Switzerland, 29-31 October 2018, Proceedings (Proceedings of Machine Learning Research, Vol. 87)*. PMLR, 651–673.

[15] B. Ravi Kiran, Ibrahim Sobh, Victor Talpaert, Patrick Mannion, Ahmad A. Al Sallab, Senthil Kumar Yogamani, and Patrick Pérez. 2022. Deep Reinforcement Learning for Autonomous Driving: A Survey. *IEEE Trans. Intell. Transp. Syst.* 23, 6 (2022), 4909–4926.

[16] Ezgi Korkmaz. 2022. Deep Reinforcement Learning Policies Learn Shared Adversarial Features across MDPs. In *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Virtual Event, February 22 - March 1, 2022*. AAAI Press, 7229–7238. https://ojs.aaai.org/index.php/AAAI/article/view/20684

[17] Yuanlong Li, Yonggang Wen, Dacheng Tao, and Kyle Guan. 2020. Transforming Cooling Optimization for Green Data Center via Deep Reinforcement Learning. *IEEE Trans. Cybern.* 50, 5 (2020), 2002–2013.

[18] Yen-Chen Lin, Ming-Yu Liu, Min Sun, and Jia-Bin Huang. 2017. Detecting Adversarial Attacks on Neural Network Policies with Visual Foresight. *CoRR* abs/1710.00814 (2017). arXiv:1710.00814 http://arxiv.org/abs/1710.00814

[19] Hong Liu, Rongrong Ji, Jie Li, Baochang Zhang, Yue Gao, Yongjian Wu, and Feiyue Huang. 2019. Universal Adversarial Perturbation via Prior Driven Uncertainty Approximation. In *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*. IEEE, 2941–2949.

[20] Jian Liu, Rui Zhang, Sebastian Szyller, Kui Ren, and N. Asokan. 2023. False Claims against Model Ownership Resolution. *CoRR* abs/2304.06607 (2023). https://doi.org/10.48550/arXiv.2304.06607 arXiv:2304.06607

[21] Sofiane Lounici, Mohamed Njeh, Orhan Ermis, Melek Önen, and Slim Trabelsi. 2021. Yes We can: Watermarking Machine Learning Models beyond Classification. In *2021 IEEE 34th Computer Security Foundations Symposium (CSF)*. IEEE, IEEE, Dubrovnik, 1–14.

[22] Nils Lukas, Edward Jiang, Xinda Li, and Florian Kerschbaum. 2022. Sok: How Robust is Image Classification Deep Neural Network Watermarking?. In *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, San Francisco, 787–804.

[23] Nils Lukas, Yuxuan Zhang, and Florian Kerschbaum. 2021. Deep Neural Network Fingerprinting by Conferrable Adversarial Examples. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net. https://openreview.net/forum?id=VqzVhqxkjH1

[24] Daniel J. Mankowitz, Nir Levine, Rae Jeong, Abbas Abdolmaleki, Jost Tobias Springenberg, Yuanyuan Shi, Jackie Kay, Todd Hester, Timothy A. Mann, and Martin A. Riedmiller. 2020. Robust Reinforcement Learning for Continuous Control with Model Misspecification. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

[25] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous Methods for Deep Reinforcement Learning. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016 (JMLR Workshop and Conference Proceedings, Vol. 48)*. JMLR.org, 1928–1937.

[26] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin A. Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529–533. https://doi.org/10.1038/nature14236

[27] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. 2017. Universal Adversarial Perturbations. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. IEEE, Honolulu, 1765–1773.

[28] Konda Reddy Mopuri, Utkarsh Ojha, Utsav Garg, and R. Venkatesh Babu. 2018. NAG: Network for Adversary Generation. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*. Computer Vision Foundation / IEEE Computer Society, 742–751.

[29] Tuomas P. Oikarinen, Wang Zhang, Alexandre Megretski, Luca Daniel, and Tsui-Wei Weng. 2021. Robust Deep Reinforcement Learning through Adversarial Loss. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan (Eds.). 26156–26167.

[30] Xinlei Pan, Chaowei Xiao, Warren He, Shuang Yang, Jian Peng, Mingjie Sun, Mingyan Liu, Bo Li, and Dawn Song. 2022. Characterizing Attacks on Deep Reinforcement Learning. In *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems* (Virtual Event, New Zealand) *(AAMAS '22)*. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 1010–1018.

[31] Zirui Peng, Shaofeng Li, Guoxing Chen, Cheng Zhang, Haojin Zhu, and Minhui Xue. 2022. Fingerprinting Deep Neural Networks Globally via Universal Adversarial Perturbations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. IEEE, New Orleans, 13430–13439.

[32] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. 2014. CNN Features Off-the-Shelf: An Astounding Baseline for Recognition. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR Workshops 2014, Columbus, OH, USA, June 23-28, 2014*. IEEE Computer Society, 512–519.

[33] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. *CoRR* abs/1707.06347 (2017). arXiv:1707.06347 http://arxiv.org/abs/1707.06347

[34] Richard S. Sutton and Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA.

[35] Sebastian Szyller, Buse Gul Atli, Samuel Marchal, and N. Asokan. 2021. DAWN: Dynamic Adversarial Watermarking of Neural Networks. In *MM '21: ACM Multimedia Conference, Virtual Event, China, October 20 - 24, 2021*, Heng Tao Shen, Yueting Zhuang, John R. Smith, Yang Yang, Pablo César, Florian Metze, and Balakrishnan Prabhakaran (Eds.). ACM, 4417–4425.

[36] Buse G. A. Tekgul, Shelly Wang, Samuel Marchal, and N. Asokan. 2022. Real-Time Adversarial Perturbations Against Deep Reinforcement Learning Policies: Attacks and Defenses. In *27th European Symposium on Research in Computer Security, Copenhagen, Denmark, September 26-30, 2022, Proceedings, Part III (Lecture Notes in Computer Science, Vol. 13556)*. Springer, 384–404.

[37] Florian Tramèr, Nicolas Papernot, Ian J. Goodfellow, Dan Boneh, and Patrick D. McDaniel. 2017. The Space of Transferable Adversarial Examples. *CoRR* abs/1704.03453 (2017). arXiv:1704.03453 http://arxiv.org/abs/1704.03453

[38] Eric Wallace, Shi Feng, Nikhil Kandpal, Matt Gardner, and Sameer Singh. 2019. Universal Adversarial Triggers for Attacking and Analyzing NLP. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*.

[39] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, and Nando de Freitas. [n. d.]. Dueling Network Architectures for Deep Reinforcement Learning. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016 (JMLR Workshop and Conference Proceedings, Vol. 48)*, Maria-Florina Balcan and Kilian Q. Weinberger (Eds.). JMLR.org, 1995–2003.

[40] Tsui-Wei Weng, Krishnamurthy Dj Dvijotham, Jonathan Uesato, Kai Xiao, Sven Gowal, Robert Stanforth, and Pushmeet Kohli. 2020. Toward Evaluating Robustness of Deep Reinforcement Learning with Continuous Control. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

[41] Mengjia Yan, Christopher W. Fletcher, and Josep Torrellas. 2020. Cache Telepathy: Leveraging Shared Resource Attacks to Learn DNN Architectures. In *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, 2003–2020. https://www.usenix.org/conference/usenixsecurity20/presentation/yan

[42] Yifan Yan, Xudong Pan, Yining Wang, Mi Zhang, and Min Yang. 2022. Cracking White-box DNN Watermarks via Invariant Neuron Transforms. *CoRR* abs/2205.00199 (2022). arXiv:2205.00199 https://doi.org/10.48550/arXiv.2205.00199

[43] Tom Zahavy, Nir Ben-Zrihem, and Shie Mannor. [n. d.]. Graying the black box: Understanding DQNs. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016 (JMLR Workshop and Conference Proceedings, Vol. 48)*, Maria-Florina Balcan and Kilian Q. Weinberger (Eds.). JMLR.org, 1899–1908.

[44] Huan Zhang, Hongge Chen, Chaowei Xiao, Bo Li, Mingyan Liu, Duane S. Boning, and Cho-Jui Hsieh. 2020. Robust Deep Reinforcement Learning against Adversarial Perturbations on State Observations. (2020).

[45] Jialong Zhang, Zhongshu Gu, Jiyong Jang, Hui Wu, Marc Ph. Stoecklin, Heqing Huang, and Ian M. Molloy. 2018. Protecting Intellectual Property of Deep Neural Networks with Watermarking. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security, AsiaCCS 2018, Incheon, Republic of Korea,*

June 04-08, 2018, Jong Kim, Gail-Joon Ahn, Seungjoo Kim, Yongdae Kim, Javier López, and Taesoo Kim (Eds.). ACM, 159–172.

A APPENDIX

A.1 Training DRL Agents

To facilitate the comparison, we used the same setup to implement all DRL policies and attacks: PyTorch (version 1.4.0), NumPy (version 1.18.1), Gym (a toolkit for developing reinforcement learning algorithms, version 0.15.7) and Atari-Py (a Python interface for the Arcade Learning Environment, version 0.2.6). All experiments were carried out on a computer with 2x12 core Intel(R) Xeon(R) CPUs (32GB RAM) and NVIDIA Quadro P5000 with 16GB memory. To train DQN agents, we used a dueling Q-network architecture proposed in [39]. For training A2C and PPO agents, we choosed to implement the convolutional neural networks suggested in OpenAI Baselines[4]. In both A2C and PPO, actors and critics use the same architecture, except for the penultimate later. The hyperparameter values of each victim agent are set the same as OpenAI baselines, while slightly differ for training independent agents.

All victim and independent DRL agents trained to play Pong reach the highest score +21. The summary of agents trained to play MsPacman is presented in Table 5. In MsPacman, we deliberately chose agents with the best performance as the victim, since they have a clear business advantage over other models, thus incentivizing adversaries to apply piracy attacks against them.

A.2 Hyperparameter Selection in FLARE

The perturbation constraint ϵ directly affects the trade-off between the success of an adversarial example and its non-transferability. Therefore, we performed a grid search for ϵ and set it to an optimal value 0.05. We set the minimum fooling rate τ_δ at a high value 0.8 to ensure the universality of the adversarial mask and set the non-transferability score at 0.7. Based on these values and Equation 4, for a candidate universal adversarial mask r, the minimum action agreement of independent agents $min_{i \in I}(AA(\pi_V, \pi_i, s, r))$ should be lower than 0.125 to be chosen as a valid fingerprint. For both the Pong and MsPacman agents, we used a reduced set of actions (4 discrete actions in total). The minimum $min_{i \in I}(AA(\pi_V, \pi_i, s, r)) = 0.125$ is much lower than 0.25 (AA, if actions are randomly chosen) and satisfies the non-transferability requirement. Finally, we set $n_{episodes}$ at a high value to guarantee that a sufficient number of fingerprints is generated for efficient verification. The prescribed hyperparameter values during fingerprint generation are listed in Table 6.

A.2.1 Selection of the Number of Fingerprints. The number of fingerprints generated and used for verification affects integrity and robustness. An insufficient number of fingerprints could result in a high action agreement AA between the independent and victim (original) policies and ultimately falsely verify the ownership of the independent policies as explained in Section 4.2. On the contrary, a high number of fingerprints could give low AA between the victim policy and its modified versions, since some of the fingerprints could give a lower fooling rate in the modified policies. For that, we performed verification by changing the maximum number of

---

[4]https://github.com/openai/baselines

Table 5: Return (averaged over 10 test episodes) of the victim and independent policies trained for MsPacman. The best and worst agents for the same DRL algorithm are highlighted in green and red, respectively.

DRL Method	Victim Agent	Independent Agents				
A2C	3316.00 ± 512.72	1670.00 ± 537.27	2552.00 ± 595.66	2144.00 ± 816.58	2246.00 ± 4.90	1750.00 ± 72.66
DQN	2620.00 ± 80.62	2363.00 ± 269.26	2484.00 ± 389.67	2218.00 ± 347.84	2211.00 ± 154.24	2472.00 ± 412.74
PPO	2731.00 ± 545.50	2019.00 ± 77.13	2198.00 ± 536.35	2040.00 ± 161.43	2017.00 ± 397.57	2167.00 ± 268.52

Table 6: Hyperparameters used in fingerprint generation

Parameter	Value	Definition
ϵ	0.05	l_∞ constraint on the perturbation r
τ_{nts}	0.7	minimum non-transferability score of r
τ_δ	0.8	minimum fooling rate of r on a dataset
$n_{episodes}$	1000	maximum number of training episodes to generate/collect fingerprints

Table 7: Total number of trials (i.e., episodes) required for obtaining the fingerprint list during the fingerprint generation phase (2nd column), and the average ratio of adversarial states that includes the fingerprint to the total number of states observed for the same episode (3rd column) during verification. The ratio is averaged over 10 verification episodes for each victim agent.

	# of trials in generation	(# of adversarial states)/(# of states) in verification
Pong, A2C	34	0.02 ± 0.00
Pong, DQN	46	0.02 ± 0.00
Pong, PPO	14	0.02 ± 0.00
MsPacman, A2C	110	0.04 ± 0.01
MsPacman, DQN	38	0.05 ± 0.01
MsPacman, PPO	10	0.05 ± 0.01

fingerprints used for fingerprint generation. As demonstrated in Figure 4, the number of fingerprints does not affect the return during verification, but a sufficient number of fingerprints (around 5) are needed to achieve high *AA* to provide high confidence for the final decision. We set the number of fingerprints at 10 to satisfy the effectiveness and robustness requirements simultaneously.

A.2.2 Selection of the Window Size. In addition to the number of fingerprints, the decision on window size is important. If the window size is large, then the return during verification decreases and the agent can perform poorly. Figure 5 illustrates the effect of window size on return and *AA* during verification for Pong DQN agents. Although *AA* does not change significantly with larger window sizes, there is a steady decline in return. Based on this result, the window size can be set to 40 or even less, but we set it to 40 after performing the same analysis for all agents and observing the change in return.

A.2.3 Computation costs of FLARE. Based on the hyperparameters chosen in our experimental setup, we computed the number of trials (i.e episodes) required to generate the fingerprint list and presented them in Table 7. The required number of trials is less than

Figure 4: The effect of the number of fingerprints on the return during verification and *AA* averaged over 10 verification episodes.

Figure 5: The effect of window size on the return during verification and *AA* averaged over 10 verification episodes.

50 in almost all cases, except MsPacman. We found that this exception occurs due to the high *AA* generated for some independent policies used during the fingerprint generation phase. To generate each $r_{candidate}$ (see Algorithm 1, line 4), FLARE randomly selects 100 states and computes the average gradient using those states. During verification, based on the window size (40) and the number of fingerprints (10), the suspected models are queried 400 times in

Figure 6: The effect of the threshold for individual finger-print's decision on the verification. Results are computed over 10 fingerprints used for the verification of victim policies, randomly selected independent policies and fine-tuned policies.

total with the additional fingerprint. Table 7 also shows the average ratio of states with an additional fingerprint to the total number of states observed during verification episodes. Based on these results, we confirm that verification episodes include only a small number of states (up to 5%) with the additional fingerprint.

A.3   Receiver Operating Characteristic of FLARE

Figure 6 shows the receiver operation characteristic (ROC) curve produced by the verification results of individual fingerprints over multiple thresholds τ_{AA}, where the i-th fingerprint votes "stolen" when $AA_i \geq \tau_{AA}$. For each victim policy, we calculated true positive and false positive rates (TPR and FPR) on 10 fingerprints that are used to verify the victim policy itself, 3 randomly selected independent policies, and 3 fine-tuned versions of the victim policy incurring a small impact on utility. We found the optimal τ_{AA} that maximizes TPR and minimizes FPR to be 0.5 and 0.68 in Pong and MsPacman, respectively. We set the threshold value at 0.5 in all our experiments, but it would be beneficial to analyze the ROC for each environment, as the choice of τ_{AA} affects the overall effectiveness and integrity of FLARE.

A.4   Impact of Verification

As discussed in Section 3.1, utility is not a necessary requirement in fingerprinting methods, as fingerprints typically trigger abnormal behavior. However, the impact on agent performance is still important, since verification might also be carried out in a stealthy way to avoid raising any suspicion. Moreover, if the agent fails to perform the task quickly during verification, then the collected information may not be sufficient to correctly calculate the action agreement AA. Therefore, we computed the impact of verification on agent performance and summarized the results in Table 8. In Pong, the impact is almost zero, while we experienced an average impact of 0.22 in MsPacman agents due to the high complexity

Table 8: Impact of verification (averaged over 10 verification episodes) on victim agent performance.

DRL Method	Pong	MsPacman
A2C	0.02 ± 0.02	0.20 ± 0.21
DQN	0.01 ± 0.02	0.28 ± 0.19
PPO	0.02 ± 0.02	0.18 ± 0.28

of the game. The impact in MsPacman can be further improved by adding fingerprints in non-critical states that do not affect the return if the agent replaces one action with another.