Fooling Edge Computation Offloading via Stealthy Interference Attack

Letian Zhang, Jie Xu
Department of Electrical and Computer Engineering, University of Miami, FL, USA

2020 IEEE/ACM Symposium on Edge Computing (SEC) | 978-1-7281-5943-0/20/$31.00 ©2020 IEEE | DOI: 10.1109/SEC50012.2020.00062

***Abstract*—There is a growing interest in developing deep learning methods to solve many resource management problems in wireless edge computing systems where model-based designs are infeasible. While deep learning is known to be vulnerable to adversarial example attacks, the security risk of learning-based designs in the context of edge computing is not well understood. In this paper, we propose and study a new adversarial example attack, called stealthy interference attack (SIA), in deep reinforcement learning (DRL)-based edge computation offloading systems. In SIA, the attacker exerts a carefully determined level of interference signal to change the input states of the DRL-based policy, thereby fooling the mobile device in selecting a target and compromised edge server for computation offloading while evading detection. Simulation results demonstrate the effectiveness of SIA, and show that our algorithm outperforms existing adversarial machine learning algorithms in terms of a higher attack success probability and a lower power consumption.**

I. Introduction

Edge computing is an emerging computing paradigm that enables in-situ processing of computation workloads locally at the network edge without moving them to the cloud, achieving lower latency, better privacy protection and reduced traffic burden on the core network. A central research problem in edge computing is how mobile devices' computation tasks should be offloaded to the edge servers. As the wireless network becomes increasingly complex and future applications pose diverse requirements, there is a growing trend to employ machine learning, especially deep learning, to address many resource management and scheduling problems in wireless edge networks, where model-based designs become infeasible [1]–[3]. For example, a particular machine learning method, namely deep reinforcement learning (DRL), has been widely adopted to solve computation offloading problems in edge computing for various design objectives [3]–[5], as it is able to deal with large state and action spaces and capture the temporal dependency of offloading decisions.

Although deep learning has achieved a huge success in various application domains, new security risks have also been exposed. It has been shown in many existing works [6]–[8] that adversarial examples can be crafted to fool machine vision systems to make wrong decisions. Such adversarial example attacks are possible because a minor modification in the input data to the deep learning algorithm may lead to a dramatic change in the output. While a significant amount of effort has been dedicated to studying adversarial example attacks in areas like machine vision, speech recognition and computer gaming, the understanding of adversarial example attacks in learning-based wireless edge system designs is very limited.

In this paper, we study a new type of adversarial example attack, called stealthy interference attack (SIA), in DRL-based edge computation offloading systems. The goal of the attacker is to fool the mobile device in selecting a target and compromised edge server for computation offloading, thereby stealing the mobile device's private data. Unlike images in machine vision, the adversarial input to the DRL-based computation offloading policy is crafted by carefully exerting an appropriate level of interference signal. Unlike conventional jamming attacks in wireless networks, the interference power is made as low as possible to evade detection by the system.

Specifically, we consider a stochastic edge computation offloading system where a mobile device decides which edge server to offload the computation task to, considering time-varying system dynamics and handover costs. We then formulate this problem as a Markov decision problem (MDP) and design a double deep Q-network (DDQN) [9] to learn the optimal computation offloading policy to minimize the long-term offloading cost. Based on the obtained DRL policy, we design the stealthy interference attack to fool the mobile device in offloading the computation task to the target edge server and optimize the amount of interference power exerted by the attacker. Simulation results are provided to demonstrate the effectiveness of SIA and show that our algorithm outperforms existing adversarial machine learning algorithms in terms of a higher attack success probability and a lower interference power consumption.

II. DRL-based Edge Computation Offloading

We consider an edge computation offloading problem of a representative mobile device. Because of its limited computing power, the mobile device has to offload its application data to an edge device for processing. We assume that there are a set $\mathcal{M} = \{1, 2, \ldots, M\}$ of candidate edge servers (ESs) in the transmission range of the mobile device, which, for example, can be co-located with base stations. We adopt a discrete-time model for this MEC system, where time is divided into time slots of equal length κ_0, which are indexed by $t = 1, 2, ...$

A. System model

In each time slot t, the mobile device generates a computation task $r_t = (\mu_t, \nu_t)$, where μ_t is the data size of the task and

This work is supported in part by the Army Research Office under award W911NF-18-1-0343 and by the National Science Foundation under awards 2029858, 2033681 and 2006630.

978-1-7281-5943-0/20/$31.00 ©2020 IEEE
DOI 10.1109/SEC50012.2020.00062

ν_t is the required total number of CPU cycles to accomplish the task. We assume that μ_t is bounded in $[\mu_{min}, \mu_{max}]$ and ν_t is bounded in $[\nu_{min}, \nu_{max}]$. We consider delay-sensitive applications where the execution time of the computation task r_t has to be no longer than the time slot length κ_0. At the beginning of each time slot t, the mobile device has to take an offloading control action $c_t \in \{\{0\} \cup \mathcal{M}\}$ to decide which ES to offload the computation task to. The mobile device is allowed to process the task locally without offloading, in which case $c_0 = 0$. Otherwise, it picks one of the ESs in $\mathcal{M}$.

Local processing: The energy consumption for processing the task on the mobile device itself during the time slot t (i.e. $c_t = 0$) can be written as

$$E_t^l = f^2 \cdot e \cdot \nu_t \tag{1}$$

where f is the CPU frequency of mobile device, e is the effective switched capacitance that depends on chip architecture of the mobile device. With the allocated CPU frequency f, the computation time of local task execution is given by

$$d_t^c = \nu_t / f \tag{2}$$

Edge offloading: The transmission rate between the mobile device and ES $m \in \mathcal{M}$ can be written as

$$\tau_t^m = W \cdot \log_2 \left(1 + \frac{g_t^m \cdot p^{tx}}{N + \alpha}\right) \tag{3}$$

where W is the bandwidth allocated to the MEC system, g_t^m is the channel state between the mobile device and ES m in time slot t, p^{tx} is the transmission power, N is the noise power and α is the background interference power. Therefore $\psi_t^m = g_t^m p^{tx}/(N+\alpha)$ is the signal-to-interference-plus-noise ratio (SINR). We assume that the SINR comes from a continuous space $\mathcal{B} = [\psi_{min}, \psi_{max}]$, where ψ_{min} and ψ_{max} are the minimum and maximum possible SINRs, respectively. Thus, the transmission delay and the transmission energy consumption are given as

$$d_t^{tx} = \mu_t / \tau_t^m, \quad E_t^{tx} = d_t^{tx} \cdot p^{tx} \tag{4}$$

Handover: We assume that in each time slot t, the mobile device can only connect with one ES. Let ρ_t denote the ES that the mobile device is connected with in time slot t, which evolves depending on the current time slot offloading decision c_t and the previous time slot connection ρ_{t-1} as follows:

$$\rho_t = m \cdot \mathbf{1}_{\{\{c_t = m\} \vee \{\{\rho_{t-1} = m\} \wedge \{c_t = 0\}\}\}} \tag{5}$$

where the symbols $\vee$ and $\wedge$ mean logic OR and logic AND, respectively, and $\mathbf{1}_{\{*\}}$ is the indicator function. Specifically, if the mobile device chooses to offload the computation task to ES m, then the connected ES must also be m. If the mobile device chooses to process the computation task locally, then the connected ES remains the same as that in the previous time slot. In the case that the connected ESs are different between two consecutive time slots, a handover occurs and an additional delay cost is incurred, which is denoted by

$$d_t^h = \eta \cdot \mathbf{1}_{\{\rho_{t-1} \neq \rho_t\}} \tag{6}$$

Thus, the total task execution delay d_t and energy consumption E_t are as follows:

$$d_t = \begin{cases} d_t^c & if \quad c_t = 0 \\ d_t^h + d_t^{tx} & if \quad c_t \in \mathcal{M} \end{cases} \tag{7}$$

$$E_t = \begin{cases} E_t^l & if \quad c_t = 0 \\ E_t^{tx} & if \quad c_t \in \mathcal{M} \end{cases} \tag{8}$$

Task failure: Since we focus on delay-sensitive tasks, task execution must be finished by the end of time slot t. This may be because (1) the computation task is offloaded to some ES but the result cannot return before the deadline or (2) the computation task is locally processed but the mobile device cannot finish processing in time. In both cases, the task fails and is dropped and hence, the mobile device incurs a failure penalty, which is denoted by

$$\phi_t = \mathbf{1}_{\{d_t > \kappa_0\}} \tag{9}$$

B. MDP Formulation

We formulate the edge computation offloading problem as a Markov decision process (MDP) problem and derive the optimal offloading policy based on DDQN.

State: The state of the mobile device can be described by $s_t = (r_t, \rho_t, \Psi_t) \in \mathcal{S}$, which is observed at the beginning of each time slot. Among the three state elements, the computation task r_t and the SINR $\Psi_t = \{\psi_t^m\}_{m \in \mathcal{M}}$ are exogenous states independent of computation offloading decision c_t, while the connection state ρ_t evolves depending on c_t. In the MDP formulation, we assume that r_t and Ψ_t evolve as Markov chains, but this Markovian assumption does not need to hold when applying our algorithm in practice. Importantly, the mobile device has no a priori knowledge about how these system states evolve.

Action: Based on the current state s_t observed at the beginning of time slot t, an offloading action $a_t \in \mathcal{A}$ is taken where $\mathcal{A} = \{\{0\} \cup \mathcal{M}\}$.

Cost: According to our system model, different actions of the mobile device can cause the different execution delays and energy consumptions. The mobile device aims to minimize the computation cost consisting of both the delay and the energy consumption, while satisfying the delay constraint. Therefore, we define a cost function $u(s_t, a_t)$ in time slot t as

$$u(s_t, a_t) = \omega_1 \cdot min\{d_t, \kappa_0\} + \omega_2 \cdot \phi_t + \omega_3 \cdot E_t \tag{10}$$

where $min\{d_t, \kappa_0\}$ is the task execution delay, ϕ_t is the task failure penalty, E_t is the energy consumption. ω_1, ω_2, ω_3 are the weights trading-off different elements of the cost function.

Policy: A computation offloading policy is a mapping: $\pi : \mathcal{S} \rightarrow \mathcal{A}$. We focus on optimizing the policy to minimize the mobile device's expected long-term cost, which is defined as the expectation of the discounted sum of the mobile device's one-slot cost.

$$V(s_0, \pi) = \mathbb{E}\left(\sum_{t=1}^{\infty} \zeta^t u(s_t, a_t) | s_0)\right) \tag{11}$$

where s_0 is the initial mobile device state, and $\zeta \in [0, 1)$ is a constant discount factor, which models the fact that a

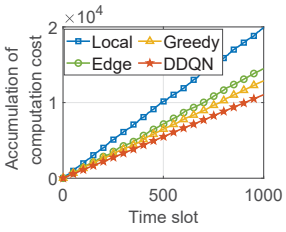

Fig. 1. Average accumulation of computation cost.

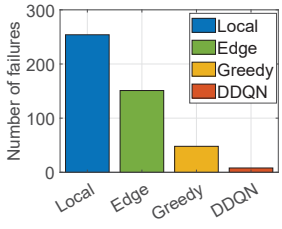

Fig. 2. Average numbers of task failures.

higher weight is put on the current cost than the future cost. The mobile device aims to design an optimal task offloading policy π^* that minimizes the expected long-term computation cost $V(s_0, \pi)$, for any given initial state $s_0 \in \mathcal{S}$, which can be formally formulated as follow

$$\pi^* = \arg\min_{\pi} V(s_0, \pi) \tag{12}$$

C. DDQN-based Policy Design

We use DDQN [9], a state-of-the-art DRL algorithm, to learn the optimal task offloading policy. It includes two neural networks, namely the generated Q-network and the target Q-network. The generated Q-network is utilized to compute the Q-value $Q(s, a, \theta)$ for each time slot while the target Q-network aims to produce the target Q-values $Q(s, a, \tilde{\theta})$ to train the parameters of the generated Q-network. We use θ and $\tilde{\theta}$ to denote the parameters of the generated Q-network and target Q-network, respectively. At the beginning of time slot t, the system state s_t is observed and a control action a_t is selected by the ϵ-greedy strategy [10], which with probability ϵ randomly selects an action and with probability $1-\epsilon$ follows the current task offloading policy. Then, with the system state s_t and the selected control action a_t, the computation cost $u(s_t, a_t)$ is realized. In the next time slot $t+1$, the new system state s_{t+1} can be observed and hence we get a state transition $h(t) = (s_t, a_t, u(s_t, a_t), s_{t+1})$ and store it into the database for experience replay. In order to train the parameters, we must collect enough transitions to provide the training samples. Specifically, the experience replay with transition size N_{re} can be represented as $H_{re} = \{h_1, h_2, ..., h_{N_{re}}\}$. To train the parameters of the DDQN model, we randomly select a mini-batch of N_{ba} transitions from N_{re} as the training samples. For each training sample $h(t) = (s_t, a_t, u(s_t, a_t), s_{t+1})$, we feed the system state s_t into the generated Q-network to compute the Q-value $Q(s_t, a_t, \theta)$. Then, we feed the system state s_{t+1} into the target Q-network to compute the Q-value $Q(s_{t+1}, a_t, \tilde{\theta})$. Following the principle of DDQN, the target value Y with this training sample can be expressed as

$$Y = u(s_t, a_t) + \zeta \cdot Q(s_{t+1}, \arg\min_{a} Q(s_{t+1}, a_t, \theta), \tilde{\theta})$$

We minimize the Huber Loss function [11] to train the parameter θ of generated Q-network. The parameters θ are copied to $\tilde{\theta}$ every T_u updates.

To illustrate the performance of the DDQN-based task offloading policy, we compare it with three baselines: **1) Local execution:** all computation tasks are executed on the mobile device; **2) Edge execution:** all computation tasks are offloaded to the ES with the best channel state; **3) Greedy execution:** In each time slot, the mobile device processes locally or offloads the task to one edge server to minimize the current time slot cost. Fig 1 shows the average cumulative computation costs and Fig 2 shows the average number of task failures of the DDQN-based policy and the three baselines. As can be seen, the DDQN-based policy outperforms the the baselines in terms of both the computation cost and the number of task failures. This demonstrates the power of DRL in solving the considered edge computation offloading problem.

III. Stealthy Interference Attack

A. Attack Model

We consider an attacker that deploys and controls L adversarial mobile devices in the MEC system, which send interference signals to the ESs, thereby changing the SINR inputs for the mobile device. We call these devices "interferer". Note the difference between "interferer" and the conventional "jammer": whereas a jammer aims to block a wireless channel to make it unusable, an interferer aims to change the SINR by adding hopefully unnoticeable interference signals.

Let the channel state between interferer l and ES m in time slot t be g_t^{lm}. With a transmission power p_t^l of interferer l, the SINR $\tilde{\psi}_t^m$ at ES m becomes

$$\tilde{\psi}_t^m = \frac{g_t^m \cdot p^{tx}}{N + \alpha + \sum_l g_t^{lm} p_t^l} \tag{13}$$

For simplicity, in this paper, we assume that $L = M$ and inferferer m can only interfere ES m. In addition, we assume that g^{mm} is normalized to 1 for all m and hence focus on the normalized interference power. With these assumptions, the SINR reduces to

$$\tilde{\psi}_t^m = \frac{g_t^m \cdot p^{tx}}{N + \alpha + p_t^m} \tag{14}$$

where p_t^m is the interference power of interferer m.

We assume that the attacker knows the optimal policy π^* of the mobile device obtained by DDQN and the true system state at the beginning of each time slot t. The goal of the attacker is to induce the mobile device to offload its computation task and data to a target ES (e.g., an ES that has been compromised by the attacker), denoted by a_Ω, by changing the state observed by the mobile. In particular, the attacker sets the transmission power of the interferer to affect the SINRs at the ESs for the mobile device. The attack is expected to be stealthy and hence the resulting SINRs should still be in a valid and rational value range. This represents a major difference from conventional jamming attacks. Formally, the attacker solves the following interference power minimization problem in every time slot t:

$$\begin{aligned} &\min \sum_{m=1}^{M} p_t^m \\ &\text{s.t. } \pi(\tilde{s}(s_t, \{p_t^m\}_{m\in\mathcal{M}})) = a_\Omega \\ &\tilde{s}(s_t, \{p_t^m\}_{m\in\mathcal{M}}) \in [s_{min}, s_t] \end{aligned} \tag{15}$$

where $\pi(\cdot)$ is the offloading policy learned by DDQN, $\tilde{s}(s_t, p_t^m)$ is the state after the SINRs are changed, and s_{min} is the lower bound of the current state s_t to ensure a stealthy attack. Clearly, if $\pi(s_t)$ is already a_Ω without changing the SINRs, then no interference attack is needed and hence the total interference power is minimized at 0.

B. Stealthy Interference Attack (SIA)

The added interference power on the SINR essentially causes a perturbation to the input to the DDQN. However, searching for the minimum perturbation is non-trivial due to the non-linear and non-convex properties of the deep neural networks. Since it is difficult to solve the power minimization problem (15) directly, we instead to solve a different problem following techniques in [12].

Let the optimal policy be given by conditional probability mass function (pmf) $\Pi^*(a|s)$. We introduce intermediate variables $\delta_t^m = \frac{\psi_t^m p_t^m}{N+\alpha+p_t^m}$ for each $m \in \mathcal{M}$. Specifically, δ_t^m is the difference in the SINR before and after adding the interference power, i.e.,

$$\delta_t^m = \psi_t^m - \tilde{\psi}_t^m = \frac{\psi_t^m p_t^m}{N + \alpha + p_t^m} \tag{16}$$

The new problem that we aim to solve is as follows:

$$\begin{aligned} &\mathcal{L}(s_t, \Pi^*, \delta_t) = \min_{\delta_t} \sum_{m=1}^{M} \delta_t^m - \xi \cdot \log \Pi^*(a_\Omega | \tilde{s}(s_t, \delta_t)) \\ &\text{s.t.} \quad \delta_t^m \in [0, \psi_t^m - \psi_{min}], \forall m \in \mathcal{M} \end{aligned} \tag{17}$$

where ξ is a positive constant. Essentially, the above problem aims to minimize δ_t^m and the cross entropy loss function, which approximates the constraint $\pi(\tilde{s}(s_t, p_t^m)) = a_\Omega$.

In order to ensure that the optimization result yields a valid interference power, δ is constrained as: $0 \le \delta_t^m \le \psi_t^m - \psi_{min}$ for all m. To handle these box constraints, we further introduce a new variable ϖ_t.

$$\delta_t^m = \frac{\psi_t^m - \psi_{min}}{2}(tanh(\varpi_t^m) + 1) \tag{18}$$

Since $-1 \le tanh(\varpi_t^m) \le 1$, it follows that $0 \le \delta_t^m \le \psi_t^{m,s} - \psi_{min}$, so the solution will automatically be valid. We can think of this approach as a smoothing of clipped gradient descent that eliminates the problem of getting stuck in extreme regions [6]. Thus, the objective function is converted to

$$\begin{aligned} \mathcal{L}(s_t, \Pi^*, \varpi_t) = \min_{\varpi_t} &\sum_{m=1}^{M} \frac{\psi_t^m - \psi_{min}}{2}(tanh(\varpi_t^m) + 1) \\ &- \xi \cdot \log \Pi^*(a_\Omega | \tilde{s}(s_t, \varpi_t)) \end{aligned} \tag{19}$$

The objective function in (19) allows us to use conventional optimization algorithms that do not support box constraints. In our implementation, we used four different solvers: 1) Powell [13]; 2) BFGS [14]; 3) Conjugate Gradient (CG) [15]; 4) Newton-Conjugate Gradient (N-CG) [16]. We found Powell to be the most effective, which finds the adversarial solution more quickly than others (see the analysis in section IV).

TABLE I
RESULTS OF SIA WITH FOUR DIFFERENT OPTIMIZATION SOLVERS

	Power consumption (W)	Success prob	Runtime (s)
BFGS	6.47×10^{-6}	42%	85.02
CG	6.98×10^{-6}	44%	240.34
N-CG	5.81×10^{-6}	36%	162.24
Powell	4.66×10^{-6}	44%	0.22

TABLE II
RESULTS OF SIA AND OTHER ADVERSARIAL ATTACKS

	Power consumption (W)	Success prob	Runtime (s)
SIA	4.66×10^{-6}	44%	0.22
Carlini [6]	4.98×10^{-6}	44%	63.52
L-BFGS [7]	4.25×10^{-6}	38%	96.01
EA [17]	5.33×10^{-6}	44%	18.10
FGS [18]	1.42×10^{-5}	37%	28.69
Naive	3.01×10^{-5}	12%	0.003
Random	7.24×10^{-7}	10%	0.01

IV. SIMULATIONS

We consider that there are $M = 5$ ESs in the network system. According to [5], the channel state Ψ and computation task arrival state r are modeled as Markov chains. We use two single hidden layers with 64 neurons for the design of the DDQN and choose $tanh$ and $relu$ as the activation functions, respectively. Other parameters used in experiments are $N_{re} = 2000$, $N_{ba} = 32$, $T_e = 500$, $T_s = 300$, $\kappa_0 = 5s$, $\mu_{max} = 65MB$, $\mu_{min} = 40MB$, $\nu_{max} = 9 \cdot 10^9$, $\nu_{min} = 7 \cdot 10^9$, $g_{max} = -2.08dB$, $g_{min} = -38.23dB$, $f = 1.7 \cdot 10^9 Hz$, $e = 4{\cdot}10^{-28}$, $\eta = 1.2s$, $W = 6{\cdot}10^5 Hz$, $p^{tx} = 2W$, $d_s = 0.1s$, $N = 10^{-8}W$, $\alpha = 0.5 \cdot 10^{-8}W$, $\omega_1 = 1$, $\omega_2 = 20$, $\omega_3 = 1$, $\zeta = 0.95$, $\xi = 20$.

A. Performance of SIA

We firstly compare different optimization solvers for solving the proposed stealthy interference attack. Then, we compare the proposed attack with other attacks. We set ES 4 as the target ES, and run each attack 50 times.

1) Determining the optimizer in SIA: Table I shows the results of SIA using four different optimization solvers, namely Powell, BFGS, CG and N-CG. From the results, Powell outperforms other three optimization solvers in terms of lower interference power, higher attack success probability and faster runtime. Therefore, we use Powell as the optimizer in SIA.

2) Comparing SIA with other attacks: Table II compares SIA with four popular adversarial ML algorithms and two representative attack benchmarks: 1) Naive attack: The attacker brings the SINRs of the non-target ESs to the minimum allowed level by sending the interference signal; 2) The attack uses random interference power to reduce SINR of the non-target ESs. As can be seen, SIA can obtain the highest successful attack probability with the lowest interference power.

B. Impact of task parameters

In this section, we investigate the impact of computation task (i.e. data size μ and demand CPU cycles ν) on the attack performance. Fig 3 shows the attack success probability for different μ and ν. For a small task data size, the attack success probability increases with the data size. This is because a larger

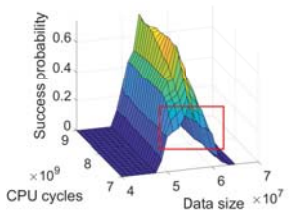

Fig. 3. Impact of arrival task on attack success probability.

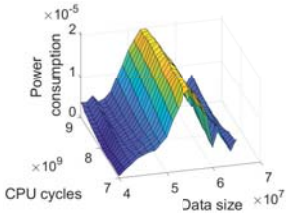

Fig. 4. Impact of arrival task on average attack power consumption.

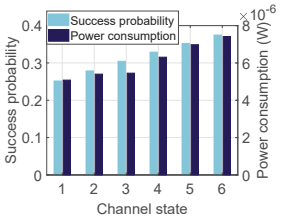

Fig. 5. Impact of channel state.

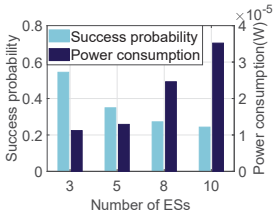

Fig. 6. Impact of number of ES.

data size means an increased transmission delay, and hence it is easier for the attack to initiate a successful attack. However, for sufficiently large data size (i.e., $\mu \geq 55MB$), the attack success probability decreases because further increasing the transmission time results in the delay requirement violation, causing the mobile device to stay with the previous slot ES. The attack success probability is less sensitive with the CPU cycle demand. The attack success probability is slightly decreased with a smaller CPU cycle demand (see the area in the red rectangle) because the mobile device is more likely to choose to execute the task locally.

Fig 4 shows the attack power consumption for different task parameters. As we can see, the required attack power consumption increases firstly and then decreases with the data size. According to the attack success probability in Fig 3, the attacker needs more interference power to modify the channel state to achieve a high attack success probability.

C. Impact of channel states

Fig 5 shows the effect of channel states by changing the target ES channel gain in $\{-38.23dB,\ -31dB,\ -23.77dB,\ -16.54dB,\ -9.31dB,\ -2.08dB\}$. As can be seen, the attack success probability and the average attack power consumption increase with the channel gain. This is because a better SINR means a lower task transmission delay, and hence the mobile device tends to choose the ES with the better channel condition. Moreover, as the SINR of the target ES improves, there are more successful attack cases even if the non-target ESs' SINRs are also high. Because more interference power is needed to bring down the high SINR of the non-target ESs, the average power consumption also increases.

D. Impact of the ES number

Fig 6 shows the influence of difference numbers of ESs on SIA. As can be seen, with more ESs, the attack success probability decreases while the average attack power consumption increases. This is because the channel state becomes more complicated with more ESs in the network, and the attacker needs more attack interference power to change more channel states to induce the mobile device to select the target ES.

V. Conclusion

In this paper, we studied a new adversarial example attack in the context of edge computation offloading and demonstrated its effectiveness in misleading the offloading decision. Future research directions include (1) black-box attacks where the attacker has no information about the mobile device's policy or input, and (2) defense strategies to protect the edge system from adversarial machine learning attacks.

References

[1] H. Ye, G. Y. Li, and B.-H. F. Juang, "Deep reinforcement learning based resource allocation for v2v communications," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 4, pp. 3163–3173, 2019.

[2] C. He, Y. Hu, Y. Chen, and B. Zeng, "Joint power allocation and channel assignment for noma with deep reinforcement learning," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 10, pp. 2200–2210, 2019.

[3] X. Chen, H. Zhang, C. Wu, S. Mao, Y. Ji, and M. Bennis, "Optimized computation offloading performance in virtual edge computing systems via deep reinforcement learning," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4005–4018, 2018.

[4] Z. Wei, B. Zhao, J. Su, and X. Lu, "Dynamic edge computation offloading for internet of things with energy harvesting: A learning method," *IEEE Internet of Things Journal*, 2018.

[5] Z. Ning, P. Dong, X. Wang, J. Rodrigues, and F. Xia, "Deep reinforcement learning for vehicular edge computing: An intelligent offloading system," *ACM Trans. Intell. Syst. Technol.*, vol. 25, p. 1, 2019.

[6] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2017, pp. 39–57.

[7] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *arXiv preprint arXiv:1312.6199*, 2013.

[8] X. Yuan, P. He, Q. Zhu, and X. Li, "Adversarial examples: Attacks and defenses for deep learning," *IEEE transactions on neural networks and learning systems*, 2019.

[9] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[10] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[11] J. H. Friedman, "Greedy function approximation: a gradient boosting machine," *Annals of statistics*, pp. 1189–1232, 2001.

[12] A. Pattanaik, Z. Tang, S. Liu, G. Bommannan, and G. Chowdhary, "Robust deep reinforcement learning with adversarial attacks," in *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 2018, pp. 2040–2042.

[13] M. J. Powell, "An efficient method for finding the minimum of a function of several variables without calculating derivatives," *The computer journal*, vol. 7, no. 2, pp. 155–162, 1964.

[14] R. Fletcher, "A new approach to variable metric algorithms," *The computer journal*, vol. 13, no. 3, pp. 317–322, 1970.

[15] R. Fletcher and C. M. Reeves, "Function minimization by conjugate gradients," *The computer journal*, vol. 7, no. 2, pp. 149–154, 1964.

[16] D. A. Knoll and D. E. Keyes, "Jacobian-free newton–krylov methods: a survey of approaches and applications," *Journal of Computational Physics*, vol. 193, no. 2, pp. 357–397, 2004.

[17] A. Nguyen, J. Yosinski, and J. Clune, "Deep neural networks are easily fooled: High confidence predictions for unrecognizable images," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 427–436.

[18] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572*, 2014.