Resilience Analysis of Deep Q-Learning Algorithms in Driving Simulations Against Cyberattacks

Godwyll Aikins, Sagar Jagtap and Weinan Gao
Department of Mechanical and Civil Engineering
College of Engineering and Science
Florida Institute of Technology
Melbourne, Florida, USA 32901
gaikins2015@my.fit.edu, sjagtap2019@my.fit.edu, wgao@fit.edu

2022 1st International Conference on AI in Cybersecurity (ICAIC) | 978-1-6654-0043-5/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICAIC53980.2022.9896968

***Abstract*—Deep reinforcement learning (DRL) has attracted attentions by researchers to complete complex tasks in engineering, such as autonomous driving, that are typically very difficult to achieve using traditional model-based approaches. With the safety being critical in self-driving vehicles and the increased reliance on vehicle connectivity, the resilience to cyberattacks has to be systematically studied. In this paper, we train a deep Q-learning based agent to drive autonomously in the CARLA simulator under various scenarios that the agent may experience during a cyberattack. Specifically, we observe the agent's behavior and performance in the presence of denial-of-service and deception attacks. The results reflect an inbuilt level of resilience to cyberattacks with the DRL methods. Comparing with conventional driving agents, deep Q-learning agents can learn to deal with uncertainty and missing information without explicitly modeling such behavior**

***Index Terms*—Resilience analysis, deep reinforcement learning, cybersecurity, autonomous vehicles, deep Q-learning**

I. Introduction

In the last decade, extensive attention and resources have been directed towards vision zero in autonomous vehicles. Among all the existing methods to achieve this ambitious goal, deep reinforcement and reinforcement learning (RL) based self-driving algorithms [5], [6], [15] have been a promising class of approaches since the DRL has proved its proficiency in other emerging applications. Deep Q-networks (DQN) [14], imitation learning, deep deterministic policy gradient (DDPG) [12], and soft actor-critic [7] are typical deep reinforcement learning (DRL) algorithms applied to autonomous vehicles.

The data from sensors, such as depth and RGB cameras, global navigation satellite systems (GNSS), inertial measurement unit (IMU), light detection and ranging (LiDAR), and radar sensors, need fused to ensure the reliability and stability of an autonomous vehicle [9]. However, the array of sensors may make autonomous vehicles susceptible and vulnerable to potential cyber threats. Therefore, in order to build safe and secure autonomous vehicles, the applied algorithms should be guaranteed to defend cyberattacks. For this purpose, Deng *et al* executed an adversarial attack that compromised the visual sensors of three state-of-the-art driving models with high accuracy [2]. It is pointed out by the authors [17] in that most of the potential cyberattacks on automated vehicles are on the sensors of the vehicle because they have very high feasibility. In [16], it is studied the significance of malicious attacks along the roadside that may change the trajectory of an autonomous vehicle. Note that although DRL may potentially improve autonomous driving performance, its resilience against cyberattacks has not been thoroughly studied. This paper will focus on the resilience of DRL algorithms against different types of cyber threats.

This work was supported in part by the U.S. National Science Foundation under Grant CMMI-2138206.

The corresponding author is Weinan Gao.

A. Cyberattacks

Typical cyberattacks on cyber physical ststems (CPSs) includes denial of service (DoS) attacks, deception attacks [13], while later can be further classified by adversarial attacks and replay attacks.

1) Denial of Service Attack: DoS attacks are strategies in which the attack impedes data distribution or restricts specific control system components to be inactivated. An example of a DoS attack in an autonomous vehicle is an adversary blocking communication of camera sensors with the control system. DoS attacks may be used to prevent cameras, LiDAR, and radar from detecting objects, roads, and safety signs. The vehicle's braking system service may fail, leading the car to stop suddenly or be unable to stop where needed. A DoS attacks usually can be characterized by two parameters, the DoS frequency, the number of times a DoS attack happens over a period, and DoS duration, a record of how long each attack lasts.

2) Deception Attack: Deception attacks, sometimes called false data attacks, are strategies in which an attacker modifies the input data used by a cyber-system. The attacker sends fraudulent data directly to the target system or decodes authorized input data and injects false data into it [22]. An example of a deception attack is a case in which an attacker takes control of a sensor and purposefully changes the readings. The attacker can provide fictitious data into the control program. Researchers in [1] were able to deceive a LiDAR-based autonomous vehicle into recognizing false obstacles to manipulate its driving judgments. The deception

attacks that this paper implemented were adversarial attacks and replay attacks.

a) Adversarial Attack: Adversarial attacks usually aim to fool machine learning algorithms to mis-classify images by altering the input image. In [20], a deep neural network is tricked by only changing one pixel on the image, it mis-classifies images on different dataset by altering one pixel. Adversarial attacks mainly started as a way to affect the results of image classification, but it is now used in broader areas.

b) Replay Attack: A replay attack is a method in which an attacker uses past recorded data to disrupt a system by feeding it as input repeatedly. When performing a replay attack, the attacker first takes a series of sensor readings. The genuine measurements are then substituted with previously recorded ones in a subsequent attack phase. As a result, the control system's performance suffers, allowing more malicious operations to go unnoticed. Due to the false data having the same signature as the actual data, replay attacks are tough to detect.

B. Deep Reinforcement Learning

Machine learning (ML) is an operation in which a computer algorithm automatically learns from its experiences to improve its performance at a specified task [4]. As a major class of ML, RL algorithms seek to train an autonomous agent to improve its performance by interacting with an environment. RL agents may learn in real-time utilizing observations gathered through real-time interactions with the environment, unlike supervised and unsupervised learning processes, which require training data to understand. The agent is not programmed to act, but this is done by reinforcing the agent's behavior. The desired actions by the agent are positively rewarded, and unwanted behavior is punished. A reward function is created to fuse the positive and negative rewards. The agent's goal is to maximize the reward over a particular duration; after a period, the agent will want to perform actions that will result in a high reward.

1) Q-Learning: Q-Learning [21] is a typical RL algorithm in which an agent aims to perform the optimal action given a state. The algorithm is broken down into three steps. First, the agent begins in a state, performs an action, and is rewarded. Second, the agent has two options for the next action: refer to the Q table and select the action with the greatest value, or perform a random action. Last but not the least, the agent updates the Q-values. A Q-table is a reference table that the agent can use to determine the appropriate course of action depending on the Q-value; see [10].

2) Deep Q-Learning: When the size of the states is a small discrete value, Q-learning is good, whereas when the states are continuous, using Q-learning to solve the problem is nearly impossible. The Q-table would have to be extremely large in continuous observation spaces. Updates on the Q-learning algorithm are also slow and do not converge rapidly. Deep Q-network was created due to the limitation of the Q-learning algorithm. One over this challenge, one can approximate the Q-function by introducing deep neural networks [19] or convolutional neural networks (CNN) [11]. In the later, deep Q-network (DQN) uses CNNs to process the observations' states by extracting features from them. Typically, the CNNs are used for classification, but in DQN, the Q-values are the resulting outputs. The network uses convolution layers to extract features from the input frames taken from the observation space. The output of the convolutional layers is flattened into a one-dimensional array which serves as the input to the fully connected layers. The fully connected layers output the Q-values associated with the defined actions. This is what forms a DQN. When training a DQN, two networks are used to create stability for the whole network: the policy and the target networks. The goal of the policy network is to find the best Q-function to approximate the most optimal policy. The target network is a clone of the policy network and is used to calculate the target value estimate y^{DQN} that is used to define the optimization criteria, which is based on the difference δ between its current Q-value estimates, and its expectation of future rewards obtained from target Q-values estimates.

$$y^{DQN} = R_t + \gamma \max_a Q(S_{t+1}, a; \theta_t^{'}) \tag{1}$$

where R_t is the reward, s is the state, a is the action. θ_t' is the weight of the target network is only updated after a certain number of steps to prevent the instability that is caused by calculating the present Q-value and predicting the future Q-value with the same network [8].

$$\delta = Q(s,a) - Q^*(s,a), \tag{2}$$

$$Q^*(s,a) = \mathbb{E}[R_{t+1} + \gamma \max_{a'} Q(s',a')] \tag{3}$$

where $Q(s,a)$ is the Q-function defined by the policy network, and $Q^*(s,a)$ is the expectation of future rewards calculated using the target network.

The agent's experiences are saved in a replay memory, and the policy network is updated by randomly selecting samples from the replay memory to boost sampling efficiency. This random selection breaks the correlation between subsequent samples. The replay memory (experience replay) allows the agent to recall and reuse previous experiences in which observed transitions are saved for a period, generally in a queue, and then randomly chosen from this memory to update the network. This method, on the other hand, merely reruns transitions at the same frequency as when they were first encountered, regardless of their importance [3].

II. Implementation and Analysis of Results

In this paper, we implement a deep Q-learning algorithm to train an agent to drive autonomously in the CARLA simulator, an open-source physics simulation platform built using Unreal Engine, which enables the simulation of realistic driving environments, making it ideal for training self-driving agents. CARLA has a variety of sensors available to use, including collision detectors, depth camera, GNSS sensor, IMU sensor,

Fig. 1. Network states

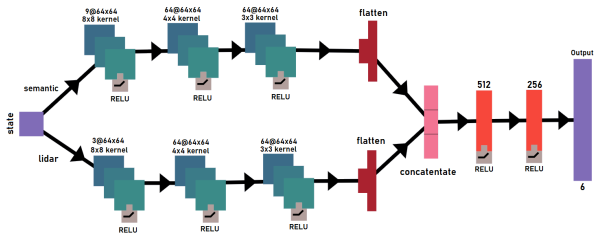

Fig. 2. Architecture of DQN

LiDAR sensor, RADAR sensor, RGB camera, RSS sensor, and optical flow camera.

A. Deep Q-Network

a) States: We use the semantic RGB image and LiDAR sensors to construct our states. The state inputs are illustrated in Fig. 1. The LiDAR sensor provides the agent with point cloud data that contribute spatial information regarding the vehicle's surroundings; see Fig. 2. The image sensor provides three-channel semantic color input showing the car's front view to give a sense of the road shape. For autonomous driving, the agent must interpret spatial and temporal information. To achieve this, we prepare the state using three consecutive simulation frames as a single environmental step and concatenate the sensor data to produce a single state matrix of size $\mathbb{R}^{12\times150\times150}$. The first three channels are 2D LiDAR point cloud images taken from consecutive time steps, and the remaining channels are the corresponding color images of the vehicle's front view.

b) Actions: We trained the DQN agent to operate the vehicle with six discrete actions. The action space depicted in Table I below describes the intended vehicle behavior [18].

TABLE I
ACTION SPACE

Actions	Descriptions
0	Slow down with 100% brake, zero steer
1	Drive forward with 60% throttle
2	Drive forward with 60% throttle and 25% left steer
3	Drive forward with 60% throttle and 25% right steer
4	Drive forward with 60% throttle and 50% left steer
5	Drive forward with 60% throttle and 50% right steer

c) Reward: Our goal with the RL agent was to design an autonomous driving agent that can keep the vehicle safe. The rewards were shaped appropriately according to this goal. Thus, the reward function needs to account for aspects of driving that are indicative of vehicle speed, distance to obstacles, and crashes, which is described as follows

$$r_t = -200r_{col} + r_{radar} + r_{speed} + r_{cont} + r_{steer} + r_{proxi}$$

where r_{col} is the reward associated with collision, r_{radar} is the reward for distance to an obstacle in front of the ego vehicle, r_{speed} is the reward associated with the car's current velocity, r_{cont} is a positive reward given to encourage long episode duration, r_{steer} is a reward set to discourage driving around in circles or turning too often, and r_{proxi} is the reward set to discourage the ego vehicle from leaving the road.

The r_{radar} is a defined by a negative step function which returns zero if the distance to an object exceeds three meters, and -1 if below to firmly discourage nearing any object. Similarly, r_{col} is -1 in case of a collision, and zero otherwise. The reward $r_{speed} = -0.0017v^2 + 0.1167v - 1$, where v is the vehicle velocity, returns small or negative rewards for very low or extremely high speeds and positive rewards otherwise. $r_{steer} = -\alpha^2$ where $\alpha \in [-1, 1]$ is the steering ratio. The proximity with non-road elements is determined using the semantic segmentation image. The semantic classification data is used to create a mask of the road surface and proximity lines are drawn over the mask to determine vicinity of non-road pixels. If the ego vehicle moves towards an obstacle detected by proximity, then a negative reward of -0.5 is given; if it drives towards a region of no detection then a reward of 0.25 is given, and if it slows down when an obstacle is ahead then a reward of 0.5 is given.

d) DQN Algorithm: Our DQN employs deep CNNs to extract high-level features from state image inputs. The network model used consists of two sets of CNN's, one to extract features from LiDAR images and the other for RGB images. Both networks consist of three convolution layers of size 64 each and kernel sizes 8, 4, and 3. The model's output is concatenated and passes through a fully connected neural network to output 6 Q-values, which provide an estimate of the optimal policy for action selection from the discrete action space; see Fig 2.

A replay buffer stores experience at each environment step for training. An experience e_t consists of the current state s_t, the action taken a_t, the future state s_{t+1}, and the reward r_t for the current state-action pair. The DQN is an off-policy training algorithm, and the replay buffer acts as a set of independent and identically distributed random variables used to train the network effectively. Replay allows us to break the correlation between consecutive samples.

Our objective is to estimate $Q^*(s, a)$ such that it is the optimal action-value function. The loss function compares the action values for the current state and action pair with the expected future rewards for the next state. We compute this expected future reward using (3) over the action values of the future state using a target network.

The loss function used in our network is defined as follows

$$L_{1,smooth} = \begin{cases} |\delta|, & \text{if } |\delta| > \alpha \\ \frac{1}{|\alpha|}\delta^2, & \text{if } |\delta| \leq \alpha \end{cases} \quad (4)$$

which provides steady gradients for large values of the error δ from (2), and more minor oscillations for small values of δ. And α is a hyper-parameter used to scale squared loss and set to unity.

Algorithm 1 Deep Q-learning Algorithm

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\hat{\theta} = \theta$
for $episode = 1 : M$ **do**
  Reset environment and get initial state s_t
  while $t = 1 : T$ **do**
    Select random action a_t with probability ϵ
    otherwise select $a_t = \arg\max_a Q^*(s, a; \theta)$
    Execute a_t in emulator and observe reward r_t and new sates s_{t+1}
    Store experience $e_t = (s_t, a_t, r_t, s_{t+1}, d)$ in D
    if Time to train **then**
    Sample mini-batch of experiences from D
    Compute targets y_j

$$y_j = \begin{cases} r_j, & \text{if episode ends at } j+1 \\ r_j + \gamma \hat{Q}(s_{j+1}, a'; \hat{\theta}), & \text{otherwise} \end{cases}$$

    Perform gradient descent step on $Loss$ with network parameters θ
    end if
    Every C steps reset $\hat{\theta} = \theta$
  end while
end for=0

e) Training Details: The simulation and training were completed in Windows 10 with CARLA 0.9.13. The deep Q-network was created using the Pytorch API. The computer the DQN was trained on had a core i7-11700K, NVIDIA 3060 12GB VRAM, and 32 GB of RAM.The network is trained for $100,000$ episodes with replay memory of length $100,000$ and batch size 128. The simulation refresh rate is set to $20FPS$. An action is selected based on an epsilon-greedy policy for a given ego vehicle state. Three simulation steps are then taken to record the next state and cumulative reward at each environment step, and this experience is stored in the replay memory. The process is repeated until the episode times out at 500 simulation steps, or if a collision is detected. The online and the target networks are synced every 3000 environment steps.

B. Cyberattacks

Autonomous driving agents are safety-critical systems, and therefore the effects of various attacks on the system must be studied. In this paper, we implement two types of attacks on our self-driving agent: denial of service, and deception.

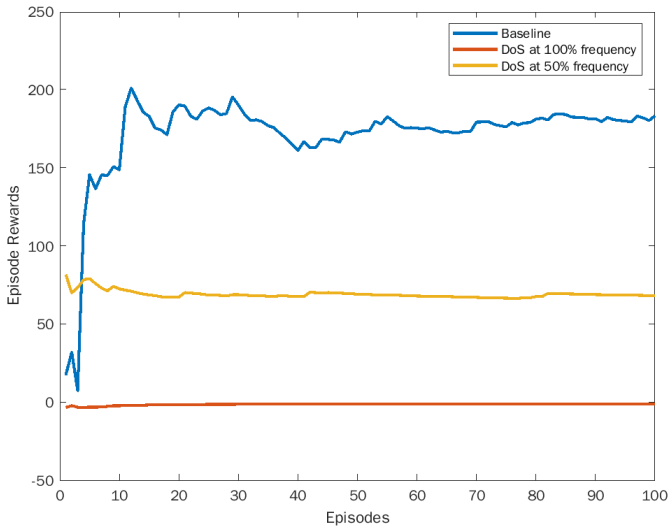

Fig. 3. Full state DoS attack

1) DoS Attacks: DoS attacks were simulated by making deliberate modifications to the flow of information in RL algorithm or attacking vehicle sensors directly to deprive the system of information critical for autonomous driving.

a) Sensor disruption: To evaluate of the DoS attacks at the sensor level, we begin by completely cutting off the sensor data stream to our network for the entire evaluation duration and observe ego vehicle behavior and rewards. This effectively simulates communication disruption between vehicle sensors and the autonomous driving agent. The experiment is repeated several times with 100% and 50% disruption intervals. The disruption intervals are simulated by interrupting sensor data at various frequencies.

Fig. 3 shows the moving average rewards observed for the above described DoS attack along with the baseline rewards. The agent seems to be quite resilient to this type of DoS attack since it appears to conclude that stopping is the optimal action when state information is missing. When the state input is completely interrupted, most of the observed actions were stop commands. Hence the agent incurs a small but steady reward penalty at each step resulting in almost a flat line reward plot. When the frequency interval is reduced to 50%, the vehicle starts exhibiting a crawl-like behavior with most of its actions being the brake command due to the empty state inputs. With this behavior, the agent incurs smaller positive rewards for movements and often completes every episode without ever crashing. Resulting in increased overall rewards compared to the 100% DoS case.

The above DoS experiments were also performed on individual sensors that are used by our DQN agent, namely LiDAR and Semantic image. First only the LiDAR signal was disrupted for the full duration of the network evaluation. Then the same experiment was repeated for the semantic image channels instead. After observing the ego vehicle behavior and the rewards presented in the Fig 4, we come to a conclusion that the agent is highly dependent on the

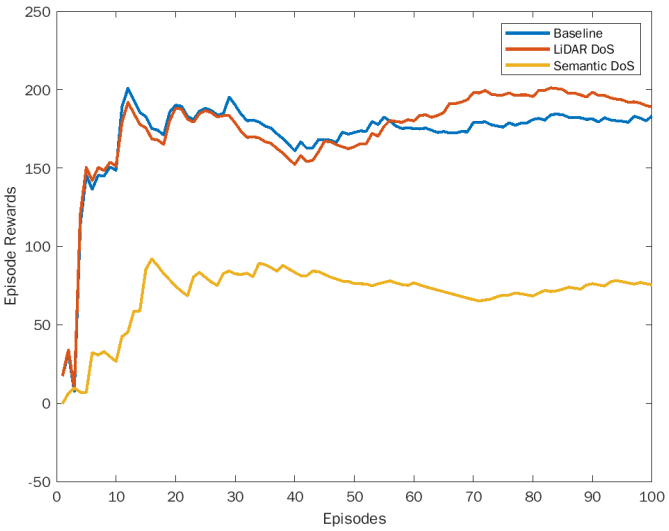

Fig. 4. DoS attack on individual sensors

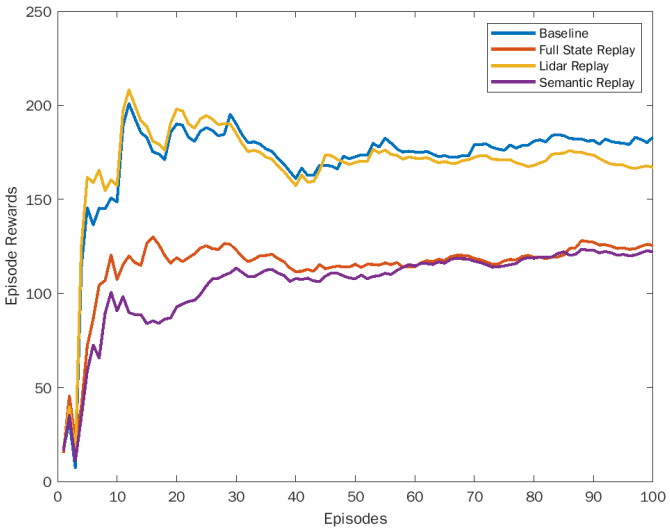

Fig. 5. Rewards for replay attacks

semantic image channels and has not yet learned to fully utilize the spatial information provided by the LiDAR. The behavior observed when only the LiDAR data was missing is almost identical to that of our baseline, whereas when the only semantic data was missing the agent resorted to crawl like behavior. Nonetheless, these experiments show that the DQN's are resilient to these types of attacks in ways that minimize the risk of accident when critical sensor data is missing.

2) Deception Attacks: To analyze the effect of a replay attack on our network, we investigate three ways our network could be attacked. In the first experiment, we input only the false LiDAR images to the LiDAR image channels of our network and note findings. We repeat the same process for the second experiment, except for the semantic image channels instead. Finally, we input the full replay buffer to the network for the last replay attack.

Fig. 5 illustrates the moving average rewards achieved by our trained network under various forms of replay attacks. For the most part, the baseline plot remains steady around 170 average reward. Based on the results of the false semantic inputs and that of the full state replay attack, the network seems to be heavily reliant on Semantic image data and does not give equal importance to the LiDAR data. This deduction is also supported by the LiDAR replay plot, whose performance is closest to the baseline.

The observed behavior of the vehicle during the simulation of the replay attacks is consistent with the plots displayed. The vehicle moved according to the fraudulent images fed as inputs. If the false semantic input showed a left turn, the vehicle should also turn left; if it is displayed states where it should drive straight, it will act accordingly. The replay images mainly were of straight roads, so the vehicle drove straight for the most part. A "stop and go" behavior was observed when only one of the input channels was false data. This behavior can be attributed to the conflicting states received by the DQN algorithm. When the false input was semantic images, the main observation was that most of the collisions were head-on. However, when the false input was LiDAR images, the bulk of the collision happened on the side of the vehicle.

III. Conclusion

In this paper, we study the resilience of deep Q learning autonomous driving algorithms through CARLA driving simulations. Different cyberattacks, such as denial-of-service and replay attacks were introduced and tested. Simulation results show that cyberattacks against the LiDAR sensor alone rarely affected the performance of the DQN; however, any attack against the semantic RGB sensor heavily affected the performance of the DQN.

References

[1] Y. Cao, C. Xiao, B. Cyr, Y. Zhou, W. Park, S. Rampazzi, Q. A. Chen, K. Fu, and Z. M. Mao, "Adversarial sensor attack on lidar-based perception in autonomous driving," *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019.

[2] Y. Deng, X. Zheng, T. Zhang, C. Chen, G. Lou, and M. Kim, "An analysis of adversarial attacks and defenses on autonomous driving models," in *2020 IEEE International Conference on Pervasive Computing and Communications (PerCom)*, 2020, pp. 1–10.

[3] A. R. Fayjie, S. Hossain, D. Oualid, and D.-J. Lee, "Driverless car: Autonomous driving using deep reinforcement learning in urban environment," in *2018 15th International Conference on Ubiquitous Robots (UR)*, 2018, pp. 896–901.

[4] H. Friji, H. Ghazzai, H. Besbes, and Y. Massoud, "A dqn-based autonomous car-following framework using rgb-d frames," in *2020 IEEE Global Conference on Artificial Intelligence and Internet of Things (GCAIoT)*, 2020, pp. 1–6.

[5] W. Gao, Z. P. Jiang, and K. Ozbay, "Data-driven adaptive optimal control of connected vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 18, no. 5, pp. 1122–1133, 2017.

[6] W. Gao, M. Mynuddin, D. C. Wunsch, and Z.-P. Jiang, "Reinforcement learning-based cooperative optimal output regulation via distributed adaptive internal model," *IEEE Transactions on Neural Networks and Learning Systems, in press*, 2021.

[7] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," *CoRR*, vol. abs/1801.01290, 2018.

[8] B. R. Kiran, I. Sobh, V. Talpaert, P. Mannion, A. A. A. Sallab, S. K. Yogamani, and P. Pérez, "Deep reinforcement learning for autonomous driving: A survey," *CoRR*, vol. abs/2002.00444, 2020.
[9] G. Kiss, "External manipulation of autonomous vehicles," in *2019 IEEE SmartWorld, Ubiquitous Intelligence Computing, Advanced Trusted Computing, Scalable Computing Communications, Cloud Big Data Computing, Internet of People and Smart City Innovation*, 2019, pp. 248–252.
[10] D. Lamba, W. H. Hsu, and M. Alsadhan, "Predictive analytics and machine learning for medical informatics: A survey of tasks and techniques," in *Machine Learning, Big Data, and IoT for Medical Informatics*, ser. Intelligent Data-Centric Systems, P. Kumar, Y. Kumar, and M. A. Tawhid, Eds. Academic Press, 2021, pp. 1–35.
[11] Y. LeCun, P. Haffner, L. Bottou, and Y. Bengio, *Object Recognition with Gradient-Based Learning*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 319–345.
[12] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.
[13] M. S. Mahmoud, M. M. Hamdan, and U. A. Baroudi, "Modeling and control of cyber-physical systems subject to cyber attacks: A survey of recent advances and challenges," *Neurocomputing*, vol. 338, pp. 101–115, 2019.
[14] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing atari with deep reinforcement learning," *CoRR*, vol. abs/1312.5602, 2013.
[15] M. Mynuddin and W. Gao, "Distributed predictive cruise control based on reinforcement learning and validation on microscopic traffic simulation," *IET Intelligent Transport Systems*, vol. 14, no. 5, pp. 270–277, 2020.
[16] N. Patel, P. Krishnamurthy, S. Garg, and F. Khorrami, "Adaptive adversarial videos on roadside billboards: Dynamically modifying trajectories of autonomous vehicles," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019, pp. 5916–5921.
[17] J. Petit and S. E. Shladover, "Potential cyberattacks on automated vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 2, pp. 546–556, 2015.
[18] N. Piazzesi, M. Hong, and A. Ceccarelli, "Attack and fault injection in self-driving agents on the carla simulator–experience report," in *International Conference on Computer Safety, Reliability, and Security*. Springer, 2021, pp. 210–225.
[19] M. Riedmiller, "Neural fitted q iteration – first experiences with a data efficient neural reinforcement learning method," in *Machine Learning: ECML*, J. Gama, R. Camacho, P. B. Brazdil, A. M. Jorge, and L. Torgo, Eds. Berlin, Heidelberg: Springer, 2005, pp. 317–328.
[20] J. Su, D. V. Vargas, and K. Sakurai, "One pixel attack for fooling deep neural networks," *IEEE Transactions on Evolutionary Computation*, vol. 23, no. 5, pp. 828–841, 2019.
[21] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3, pp. 279–292, May 1992.
[22] D. Zhang, Q.-G. Wang, G. Feng, Y. Shi, and A. V. Vasilakos, "A survey on attack detection, estimation and control of industrial cyber–physical systems," *ISA Transactions*, vol. 116, pp. 1–16, 2021.