Towards Physically Adversarial Intelligent Networks (PAINs) for Safer Self-Driving

Piyush Gupta, Graduate Student Member, IEEE, Demetris Coleman, Graduate Student Member, IEEE, and Joshua E. Siegel, Member, IEEE

Abstract—Neural networks in autonomous vehicles suffer from overfitting, poor generalizability, and untrained edge cases due to limited data availability. Researchers often synthesize randomized edge-case scenarios to assist in the training process, though simulation introduces the potential for overfitting to latent rules and features. Automating worst-case scenario generation could yield informative data for improving self-driving. To this end, we present a “Physically Adversarial Intelligent Network”, wherein self-driving vehicles interact aggressively in the CARLA simulation. We train two agents, a protagonist, and an adversary, using dueling double deep Q networks with prioritized experience replay. The coupled networks alternately seek to collide and avoid collisions such that the “defensive” avoidance algorithm increases the mean time to failure and distance traveled under non-hostile operating conditions. The trained protagonist becomes more resilient to environmental uncertainty and less prone to corner case failures resulting in collisions than the agent trained without an adversary.

Index Terms—Autonomous vehicles, adversarial reinforcement learning, physically adversarial intelligent network, dueling double deep Q network, prioritized experience replay.

I. INTRODUCTION

AUTOMATED Vehicles (AVs) are an imminent reality, and to reach mainstream adoption, AVs must ensure safe and efficient operation through intelligent decision-making. To this end, AVs must be exposed to and learn from an abundance of training data including real-world chaos [1].

Due to economical cost constraints and the risk of physical damage, certain informative scenarios cannot be captured as real-world training data. Some self-driving systems mitigate risk by avoiding high-speed operation in unfamiliar environments [2], or simulated environments may be used to allow AVs to observe atypical and infrequent experiences, providing faster-than-real-time exposure to simulated scenarios and improving real-world performance. Traditional simulations implement scenarios designed by humans or with variational tools, leading to excluded edge cases. Human behavior, maintenance issues, and logical errors lead to unpredictability. Simulations lack the entropy of a chaotic environment which is critical to effectively train a defensive self-driving car. Therefore, recent years have seen significant efforts towards generating safety-critical edge-case scenarios [3], [4]. It is critical to incorporate these scenarios in training data as the AVs’ neural networks, exposed to unseen conditions - even slight variations on training data - may behave unexpectedly, with grave consequences.

To mitigate this, we introduce a “Physically Adversarial Intelligent Network (PAIN),” pitting self-driving cars against one another to create a hostile, entropic environment. PAIN is inspired by “Generative Adversarial Networks (GANs) [5],” a means of pitting two neural networks, like those used to pilot self-driving cars [6], against one another to improve driving policies for both the attacker and the defender. Recent studies utilize GANs for self-driving applications [7], [8], [9], [10]. In these works, GANs either generate edge-case driving scenarios to test and validate self-driving cars, or generate adversarial attacks on the AV’s perception stack. In contrast, we use deep reinforcement learning (RL) to jointly train two self-driving agents: a protagonist and an adversary, where the protagonist completes an autonomous driving task and the adversary forces edge-case scenarios. This allows for the automatic generation and inclusion of edge-case scenarios in the training data to effectively train the protagonist.

Deep RL is a popular paradigm for training AVs (see [11] for an overview). End-to-end approaches such as imitation learning [12] have shown promising results. However, these RL policies may not adapt to the real world due to insufficient simulated data and limited scenario coverage, leading to untrained edge cases. Authors in [13], [14] propose robust adversarial reinforcement learning (RARL) where, a protagonist and an adversary, are pitted against one another in a two-player zero-sum game, and the adversary acts as a destabilizing disturbance to the protagonist. In [14], the two agents take turns controlling the same vehicle so that the protagonist learns to robustly navigate a simulated race track. In contrast to the RARL approach where the adversary acts as a disturbance, [15] presents multi-agent reinforcement learning...
(MARL), where the adversary is an agent competing in the same environment. Promising results were shown in [16], where a team-based hide-and-seek game led to complex behaviors, and teams learning new strategies to thwart one another.

In this letter, we pit the protagonist and the adversary against one another in the CARLA [17] simulation. The protagonist’s objective is to drive safely from a start to a goal location, while the adversary seeks to maximize the damage to the protagonist through “physical” collisions. This helps generate rare, noisy data where the protagonist learns to anticipate and avoid impending direct collisions, while the adversary learns to cause increasingly-unpredictable collisions. The resultant data is a better representative of real-world scenarios than that provided by pre-programmed or randomized simulation alone.

To ease implementation on constrained compute platforms, like those found in AVs, we train our agents using efficient Dueling Double Deep Q Network (DDDQN) [18] with prioritized experience replay [19] algorithm. We additionally utilize assisted exploration by incorporating a stochastic PID controller during exploration, and frame-skip [20] to accelerate training. Using these techniques, i.e., priors on policies [21] and frame-skip [20], are known to improve the speed and stability of the training process. We show that the protagonist trained with an adversary learns to drive defensively and exhibits a higher success rate (safely reaching the goal location), with an increase in Mean-Time-to-Failure (MTTF) and average travel distance in unseen driving scenarios than the agent trained alone. We also show that with no surrounding vehicles, adversarial training does not impact performance negatively.

The contributions of this letter are fourfold: (i) we introduce PAIN, which pits self-driving vehicles with different objectives against one another with the environment-in-the-loop, (ii) we utilize the DDDQN algorithm with prioritized experience replay to train self-driving agents in CARLA simulation, (iii) we utilize assisted exploration and frame-skip to speed up the training of the coupled agents, (iv) we show that the trained “defensive driving” agent becomes more resilient to edge cases than the agent trained without an adversary. Specifically, we show that the PAIN-trained avoidance algorithm outperforms the baseline (protagonist trained alone) in all measured performance metrics.

II. PROBLEM FORMULATION

We consider two agents, an adversary, and a protagonist, that seek to collide and avoid collisions, respectively. The agents are trained using model-free Deep RL in the CARLA simulation environment. Fig. 1(a) shows the protagonist’s 1.4 km desired route. The red curve shows the desired protagonist’s trajectory to drive safely from the start location (green circle) to the goal location (blue circle). We allow nine discrete actions \( A \in A \) for both agents, given by \( A = (T, S) \), where \( T \in \{ Cons, Acc, Dec \} \) includes constant speed, acceleration, and deceleration, respectively, and \( S \in \{ Cons, Left, Right \} \) includes constant steering, steer left, and steer right, respectively. To control the vehicle, CARLA allows the following control inputs: (i) steering \( St \in [-1, 1] \), (ii) throttle \( Th \in [0, 1] \), and (iii) brake \( Br \in [0, 1] \). For each action \( A = (T, S) \), the control update is given by:

\[
St = \begin{cases} 
Stpr, & \text{if } S = Cons, \\
\min\{Stpr + 0.2, 1\}, & \text{if } S = Right, \\
\max\{Stpr - 0.2, -1\}, & \text{if } S = Left,
\end{cases} 
\]

\[
Th = \begin{cases} 
Thpr, & \text{if } T = Cons, \\
\min\{Thpr + 0.2, 1\}, & \text{if } T = Acc, \\
\max\{Thpr - 0.2, 0\}, & \text{if } T = Dec,
\end{cases} 
\]

\[
Br = \begin{cases} 
Brpr, & \text{if } T = Cons, \\
0, & \text{if } T = Acc, \\
\max\{Brpr + 0.2, 1\}, & \text{if } T = Dec,
\end{cases} 
\]

where \( Stpr, Thpr, \) and \( Brpr \) denote the steering, throttle, and brake commands at the previous time-step, respectively. The incremental control updates (1)-(3) are designed with a discrete step-size 0.2 and ensure that the control commands remain in their desired range. Incremental, control-targeted changes avoid abrupt changes in the outputs that might violate a real vehicle’s kinematic constraints. We now provide our input representation and reward structures for the agents.

A. Input Representation

The CARLA simulation includes high-dimensional information like road markings, traffic signs, etc. While AVs require these data for safe operation, this letter aims to study the impact of an adversary on training “defensive” agents and therefore does not require considering traffic laws. Unlike a real-world AV which would utilize RGB cameras, LiDAR, RADAR, and other sensors to cover the 360° field-of-view of the vehicle, we instead utilize only the front view image, capturing 110° field-of-view as perceptive input. To simplify the high-dimensional front view, we utilize CARLA’s semantic camera to pre-segment an image. Due to recent advances in semantic segmentation [22], AVs can be equipped with such algorithms. We therefore assume the availability of such data.

We reduce the state complexity by post-processing the semantically segmented image to mask out vegetation and road markings. Figs. 1(b) and 1(c) show a protagonist’s front
view using (a) an RGB camera and (b) the semantic segmentation camera. The processed, masked image is shown in Fig. 1(d). This image is converted to a normalized gray-scale image (Fig. 1(e)) of size 100x120, which is used in a sequence of four consecutive frames (to predict vehicle motion) as a perceptive input to the network. To simplify motion computation, we assume the availability of onboard sensors describing vehicle pose and relative motion. Our augmented input state $s \in \mathcal{S}$ to the neural network consists of (i) a sequence of four segmented, masked, normalized grayscale images, (ii) vehicle motion state: $(v_{lon}, v_{lat}, \omega, a_{lon}, a_{lat})$ and, (iii) previous control commands: $(T_{thpr}, S_{pr}, B_{pr})$, where, $v_{lon}(a_{lon})$ and $v_{lat}(a_{lat})$ represents the longitudinal and lateral velocity (acceleration) of the ego vehicle, respectively, and $\omega$ represents its yaw rate. Vehicle motion state and control commands are easily accessible through common vehicle sensors such as IMUs, GNSS, encoders, etc.

B. Reward Design

**Protagonist:** The protagonist’s objective is to safely drive from a start location to a goal location (Fig. 1(a)) in the minimum time without any collisions, subject to a maximum “safe” acceleration limit. For each state $s \in \mathcal{S}$ (Section II-A) and action $A^p \in \mathcal{A}$ by the protagonist, we design the reward function $R^p(s, A^p) \in \mathbb{R}$:

$$R^p(s, A^p) = R^p_{v} - R^p_{a} - R^p_{r} - R^p_{col} - R^p_{cross}, \quad (4)$$

where $R^p_v$, $R^p_a$, $R^p_{r}$, $R^p_{col}$, $R^p_{goal}$ and $R^p_{cross}$ are the rewards based on absolute velocity $v^p$, absolute acceleration $a^p$, steering angle $s^p$, collision event, distance to goal $dis_{goal}$, and cross-track error $e^p_{cross}$ (minimum euclidean distance from protagonist location to its target trajectory), respectively. Reward terms are defined as follows:

$$R^p_v = \left\{ \begin{array}{ll}
-\frac{r_1}{v_{max}} & \text{if } v^p \leq v_{min}, \\
\frac{r_2}{v_{max}}(\cos \theta^p - \sin \theta^p) & \text{if } v_{min} \leq v^p \leq v_{max}, \\
0 & \text{if } v^p > v_{max},
\end{array} \right. \quad R^p_a = r_3 \times (a^p \geq a_{max}), \quad R^p_{r} = r_4 \times (s^p)^2, \quad R^p_{goal} = r_5 \times \left( 1 - \frac{dis_{goal}}{R_{len}} \right) + r_6 \times 1(dis_{goal} < \delta), \quad R^p_{cross} = r_7 \times 1(coll^p), \quad R^p_{cross} = r_8 \times \left( \frac{2 \times e^p_{cross}}{roadwidth} \right)^2,$$

where $r_i, i \in \{1, \ldots, 8\}$ are positive constants, $\theta^p$ is the angle of the protagonist with respect to the lane, and $1(\cdot)$ denotes the indicator function which is true when the corresponding condition (\cdot) is true. The reward $v^p \cos(\theta^p) - v^p \sin(\theta^p)$ encourages (penalizes) the protagonist’s velocity along (normal to) the lane direction. $v_{min}$, $v_{max}$, and $a_{max}$ denote minimum and maximum limits of the absolute velocity and acceleration. The penalty $-r_1$ in $R^p_v$ minimizes unnecessary stopping. $R^p_a$ and $R^p_{r}$ penalize the agent for large accelerations and over-steering, respectively, for stable and comfortable driving. $R^p_{col}$ provides a penalty for collisions determined by the CARLA collision detector. $R^p_{goal}$ provides a reward based on the distance to the goal normalized against the route length $R_{len}$, and for successfully reaching the goal point within a threshold $\delta$. $R^p_{cross}$ penalizes the vehicle based on the cross-track error from the desired path which we normalize with the half-width of the road $roadwidth$.

**Adversary:** The adversary aims to maximize its absolute acceleration by colliding with the protagonist. Due to the limited front perceptive field of both agents, we spawn the adversary facing the protagonist at some distance. We design the adversary’s objective function to encourage driving towards the protagonist (without environmental collisions), making a collision with the protagonist the top priority. For each state $s \in \mathcal{S}$ and action $A^a \in \mathcal{A}$ by the adversary, we utilize the reward function $R^a(s, A^a) \in \mathbb{R}$:

$$R^a(s, A^a) = R^a_v + R^a_{col} + R^a_{dis} - R^a_{cross} + R^a_{goal} - R^a_{st}, \quad (5)$$

where $R^a_v$, $R^a_{col}$, $R^a_{dis}$ and $R^a_{cross}$ are rewards based on the absolute velocity $v^a$, collision event, distance from the protagonist $dis_{pro}$, and cross-track error $e^a_{cross}$, respectively. To encourage driving towards the protagonist, we do online route planning from the adversary spawn location to the protagonist’s start point and utilize it to compute the cross-track error term $R^a_{cross}$, $R^a_{goal}$, and $R^a_{st}$ rewards are analogous to the protagonist’s to encourage driving towards the goal point (starting point of the protagonist) and limiting over-steering, respectively. Other terms are defined as:

$$R^a_v = \left\{ \begin{array}{ll}
\frac{r_2 v^a}{v_{max}}(\cos \theta^a - \sin \theta^a) & \text{if } 0 \leq v^a \leq v_{max}, \\
0 & \text{if } v^a > v_{max},
\end{array} \right. \quad R^a_{col} = r_{10} \times 1(coll^a = pro) - r_7 \times 1(coll^a \neq pro),$$

$$R^a_{dis} = r_9 \times \left( \frac{1 - dis_{pro}}{d} \right), \quad R^a_{cross} = r_8 \times \left( \frac{2 \times e^a_{cross}}{roadwidth} \right)^2 \times 1(e^a_{cross} > \Delta),$$

where $r_9, r_{10}, \Delta$ and $d$ are positive constants. $R^a_v$ rewards the adversary for driving at a high-velocity opposite to the protagonist as it will produce peak maximum acceleration and maximum damage potential. $R^a_{col}$ rewards the adversary for successful collision with the protagonist and penalizes it for any other collisions. $R^a_{dis}$ rewards the adversary based on its distance from the protagonist. The term $R^a_{cross}$ penalizes the adversary for a large ($>\Delta$) cross-track error, making the adversary trajectory susceptible to environmental collisions. This avoids forcing any specific trajectory on the adversary. Note that without the reward terms $R^a_{cross}$, $R^a_{goal}$ and $R^a_{st}$, the adversary learns to circle in the roadway at $v_{max}$, blocking the protagonist’s route. While such a policy will produce a high hit rate for the adversary, the generated data will not be as diverse as that from other policies.

III. PHYSICALLY ADVERSARIAL INTELLIGENT NETWORK

In RL, Deep Q-learning [23] is often used to train agents with discrete action spaces. Algorithms such as deep Q networks (DQN) [23] and double Q networks (DDQNs) [24] utilize neural networks to cope with large state spaces by estimating Q-values. These algorithms suffer from an overestimation of Q values and unstable learning.
DDDQN [18] improves upon these by estimating Q-values in a more reliable, faster, and stable manner.

**Fig. 2** shows the PAIN framework that trains coupled networks in a high-entropy environment to learn robust protagonist and adversary policies. We utilize DDDQN with prioritized experience replay for agent training. The PAIN framework includes coupled adversarial training in simulation followed by their deployment on small-scale “physical” systems [25]. While the current work focuses on the simulation study, our future work will focus on the physical deployment of the PAIN agents. We now discuss the PAIN implementation’s algorithms, architecture, and methods.

### A. Dueling Double Deep Q Network (DDDQN)

While the next action is critical in some states, it makes little difference in other states. Whereas an active state may be critical for safety when driving on a narrow curvy road, it may not be in a wide open parking lot. For states with minimal action impact, it is unnecessary to estimate each action’s value. To this end, DDDQN estimates Q-values by aggregating the state value \( V(s) \) of being at a state and the advantage \( A(s, a) \) of taking a specific action at that state. This is realized by splitting the output of the convolutional layers in a deep-Q network into two streams of fully connected layers, one for estimating the value of the state \( V(s) \) and the other for the action advantage \( A(s, a) \). The final Q-value estimate in DDDQN is obtained by aggregating estimates in an aggregation layer as follows:

\[
Q(s, a; w, \alpha, \beta) = V(s; w, \beta) + (A(s, a; w, \alpha) - \frac{1}{|A|}A(s, a'; w, \alpha)),
\]

where \( w, \alpha, \beta \) are the common network weights, advantage stream parameters, and the value stream parameters, respectively. Decoupling into two streams accelerates training by providing a more reliable estimate of Q values for each action [18]. Additionally, the network learns to identify whether a state is desirable while identifying the importance of each action in that state. DDDQN, therefore, allows for faster, stable learning with reliable Q-value estimates.

### B. Prioritized Experience Replay (PER)

PER [19] samples a batch of experiences from a memory buffer to train a network. It improves the policy learned by DQN algorithms by increasing the replay probability of experiences that have a high impact on learning. These experiences may be rare but informative. The prediction error of the Q-learning algorithm is used to assign a priority value \( p_i \) for each experience in a memory buffer, which then generates a probability \( P(i) = \frac{p_i}{\sum_{i=1}^{N} p_i} \) by normalizing it with the total priority values of the experiences held in the memory. \( \lambda \in [0, 1] \) is a hyper-parameter that adds randomness to experience selection. Sampling experiences based on \( P(i) \) tends to add bias to the training data since high-priority experiences will get selected more often. To remove this bias, PER weights experiences with importance sampling weights (IS) calculated as \( \omega_i = (\frac{1}{N^{|P|}})^{\lambda} \), where \( N \) is the number of experiences in memory, and the hyper-parameter \( \mu \in [0, 1] \) controls the impact of IS weights on learning. PER improves learning speed and policy quality when compared to uniform experience replay [19].

### C. Network Architecture

**Fig. 3** shows our DDDQN-based neural network architecture for both agents. The sequence of four stacked normalized gray-scale images of size 100x120 is passed through 3 convolution layers with filter sizes 8x8x4-s-4, 4x4x32-s-2, and 4x4x64-s-2. The output after 3 convolutions (4x5x128) is flattened and passed to two 512-dimensional fully connected layers along with the vehicle motion state and previous control commands. The two fully connected layers estimate the state value \( V(s) \) and 9-dimensional advantage vector \( A(s, a_i) \), \( i \in [1, \ldots, 9] \). Finally, the state value and advantage vector are aggregated (see (6)) through an aggregation layer to produce a 9-dimensional vector corresponding to the Q-values for all possible actions. Similar architectures have been used to train agents to play computer games [26].

### D. Assisted Exploration

Due to the high-dimensional state space for autonomous driving, training is time-intensive. Researchers utilize techniques like imitation and transfer learning to speed up the training process [25]. We introduce assisted exploration utilizing a stochastic PID controller during exploration to speed up the training process. Specifically, we tune a PID controller to follow a target path and utilize it to generate preconditioning data along with random exploration during training. Unlike the epsilon greedy approach, where random actions are chosen during exploration, we choose exploration actions based on
Algorithm 1 Assisted Exploration Using the PID Controller

Input: Action set A, PID action $A_{PID} \in A$. Exploration probability $\epsilon$, state $s$, DQN network Net, Previous Control command $(St, Th, Br)$

Output: Action $A$ and Control command $(St', Th', Br')$

1: procedure GETACTION
2: Sample a random number $a \in [0, 1]$
3: if $a < \epsilon$ then
4: $PID$ probability $P_{PID} \leftarrow 0.75\epsilon$
5: Sample a random number $a'_{PID} \in [0, 1]$
6: if $a'_{PID} < P_{PID}$ then
7: $Action A \leftarrow A_{PID}$
8: else
9: Sample a random action $A \in A$
10: else
11: Obtain estimated $Q$ values $Q \leftarrow Net(s)$
12: $Action A \leftarrow \arg\max_{a}Q(s, a)$
13: $(St, Th, Br) \leftarrow CONT((St, Th, Br, Br), A)$
14: return $A, (St, Th, Br)$

the PID controller with probability $P_{PID}$ and random actions with probability $1 - P_{PID}$. To reduce the effect of the PID controller over time, we decrease $P_{PID}$ with the training steps. Algorithm 1 provides pseudo-code for the assisted exploration strategy, where the function $CONT((St, Th, Br, Br), A)$ calculates the control input using (1)-(3). We exponentially decrease the exploration probability $\epsilon$ with training steps. Since the output of the PID controller does not necessarily satisfy (1)-(3), the input PID action $A_{PID} \in A$ is calculated by mapping the output of the PID controller and previous control command to the corresponding action in $A$. Without assisted exploration, the untrained agent may not explore enough of the state space to learn an effective policy from random actions.

E. Frame-Skip

We utilize frame-skip to accelerate training, in which the action of the agent is kept unchanged for $k$ consecutive frames. This technique reduces training complexity and leads to stable policies. While training the protagonist, we change $k$ from one to three at pre-determined intervals. Initially, actions are chosen from the PID controller with high probability. Utilizing frame skip and PID with large $k$ causes error accumulation and unstable agent motion, leading to frequent collisions. Therefore, we increase $k$ from one to three after a significant reduction in the exploration probability.

IV. EXPERIMENTAL EVALUATION

We first train the protagonist alone to learn a safe policy to reach the goal location without collisions. This model serves as a baseline for the protagonist. Once the baseline learns to safely drive the route, we start our coupled training with the adversary. We clone the baseline policy as the initial policy for both the adversary and the protagonist. During adversarial training, the performance of two agents constantly fluctuates against each other as they learn new ways to oppose each other’s policies. Therefore, we train two protagonist models with a different number of training episodes; Model 1 (243K episodes) and Model 2 (353K episodes).

We utilize success rate, mean-time-to-failure (MTTF), and mean distance traveled (normalized with total route length) as performance metrics for the protagonist. In scenarios where the protagonist successfully completes the route, MTTF is equivalent to completion time. The success rates are recorded at four checkpoints: 25%, 50%, 75%, and 95% of route length. In case of failure through collision, we record the event’s collision intensity (CI).

We evaluate the protagonist’s performance in four scenarios\(^1\): (1) without any surrounding vehicles; (2) with 50 nearby vehicles driving in CARLA autopilot mode; (3) 5 static vehicles obstructing the protagonist’s route; and (4) against the adversary. We conduct 100 trials for each scenario and compare the performance with the baseline model.

Scenario 1 (No Surrounding Vehicles): We evaluate the protagonist’s performance in an environment with no other vehicles. This is the baseline agent’s training environment, a useful scenario to evaluate whether the PAIN degraded baseline performance in safe environments (e.g., by learning over-conservative policies such as slow driving, or a bang-bang control strategy).

All tested models were able to complete the track with a 100% success rate and therefore traveled the same distance without any collisions. The average reward of all the models was similar which indicates that the performance of the PAIN’s protagonist does not deteriorate in safe environments.

Scenario 2 (Dynamic Vehicles in Autopilot Mode): This scenario emulates typical driving with surrounding vehicles. We randomly spawn 50 vehicles in the environment driving in autopilot mode. Due to the large environment, the agent might only interact with a few vehicles (5-15) along its route. This scenario is challenging as the protagonist has never seen more than one vehicle (adversary) during training.

Some rear-end collisions occurred, which are unavoidable due to the limited perceptive field (110° field-of-view) for all models’ input. Table I compares the performance of the protagonist in Scenario 2 with the baseline model. Protagonists trained through the PAIN framework outperform the baseline in all performance metrics, and Model 2 outperforms Model 1 in all metrics, indicating that the PAIN framework will continue to improve over long timescales.

Scenario 3 (Static Vehicles Obstructing Protagonist’s Route): In this scenario, we obstruct the protagonist’s trajectory by spawning 5 static vehicles along its route.

\(^1\)Video available at https://youtu.be/2x3-QQXCr1o.
This provides edge cases where the static vehicles block the road and emulate real-world accidents, roadkill, etc. It is a challenging scenario for the protagonist because: (i) multiple vehicles were not encountered in training, (ii) static vehicles were not in the training set, and (iii) the route blockage created by static vehicles forces the agent to deviate from its typical trajectory and attempt high-risk trajectories leading to unseen forward-facing camera data. Table II compares the performance of the protagonist models in Scenario 3 with the baseline model. The protagonist models trained through the PAIN framework outperform the baseline in all performance metrics.

**Scenario 4 (Against Adversary):** This scenario pits the protagonist against the trained adversary. The adversary was spawned facing the protagonist at a random distance in front of it. We evaluate the hit rate of the adversary, characterized as a successful adversary–protagonist collision of any force. In 100 trials, the hit rates of the adversary against the baseline, Model 1, and Model 2 were 22%, 34%, and 28%, respectively. Since the adversary is trained against the protagonist, it has a higher hit rate against these models than the baseline suggesting that, like the protagonist, the adversary improves over episodes. In practice, we assume that the vehicle collisions are incidental, and therefore the protagonist will be better off in day-to-day driving, no matter how effective the adversary becomes.

### V. Conclusion and Future Directions

We proposed a “physically adversarial intelligent network (PAIN)” pitting multiple AVs against one another with the environment-in-the-loop. The coupled networks attempt to find faults in one another which improves the performance of both the protagonist and the adversary. We show that the protagonist trained with the adversary outperforms the baseline model in all performance metrics. The presence of the adversary leads to more robust obstacle avoidance policies for the protagonist as well as provides edge case training scenarios that are difficult to pre-program.

In our future work, we plan to add multiple cameras and other sensors to improve the perceptive field, utilizing transfer learning based on our current network. Perhaps most important is the planned deployment of PAIN on small-scale physical vehicles [25]. Hundreds of small-scale vehicles can be built for the price of one AV, leading to a happy balance of data collection cost, diversity, and speed.

### References

[1] A. C. Madrigal, *Inside Waymo’s Secret World for Training Self-Driving Cars*, vol. 23, Boston, MA, USA: Atlantic, 2017.

[2] G. Kahn, A. Villaflor, V. Pong, P. Abbeel, and S. Levine, “Uncertainty-aware reinforcement learning for collision avoidance,” 2017, arXiv:1702.01182.

[3] Y. Abeyasingiroonawardena, F. Shkurti, and G. Dudek, “Generating adversarial driving scenarios in high-fidelity simulators,” in *Proc. Int. Conf. Robot. Autom.*, 2019, pp. 8271–8277.

[4] M. Prisula, A. Pirinen, C. Padurarut, and C. Sminchisescu, “Generating scenarios with diverse pedestrian behaviors for autonomous vehicle testing,” in *Proc. Conf. Robot. Learn.*, 2022, pp. 1247–1258.

[5] I. Goodfellow et al., “Generative adversarial nets,” in *Proc. Adv. Neural Inf. Process. Syst.*, 2014, pp. 2672–2680.

[6] A. Ghosh, B. Bhattacharya, and S. B. R. Chowdhury, “SAD-GAN: Synthetic autonomous driving using generative adversarial networks,” 2016, arXiv:1611.08788.

[7] A. Demetriou, H. Allsväg, S. Rahrovani, and M. H. Chehreghani, “Generation of driving scenario trajectories with generative adversarial networks,” in *Proc. 23rd Int. Conf. Intell. Transp. Syst. (ITSC)*, 2020, pp. 1–6.

[8] A. Hamdi, M. Müller, and B. Ghahem, “SADA: Semantic adversarial diagnostic attacks for autonomous applications,” in *Proc. AAAI Conf. Intell. Artif. Intell.*, vol. 34, 2020, pp. 10901–10908.

[9] W. Ding, B. Chen, M. Xu, and D. Zhao, “Learning to collide: An adaptive safety-critical scenarios generating method,” in *Proc. Int. Conf. Intell. Robots Syst. (IROS)*, 2020, pp. 2243–2250.

[10] M. Zhang, Y. Zhang, L. Zhang, C. Liu, and S. Khursheed, “DeepRoad: GAN-based metamorphic testing and input validation framework for autonomous driving systems,” in *Proc. 33rd ACM Int. Conf. Automated Softw. Eng. (ASE)*, 2018, pp. 132–142.

[11] A. E. Sallab, M. Abdou, E. Perot, and S. Yogamani, “Deep reinforcement learning framework for autonomous driving,” *Electron. Imag.*, vol. 19, pp. 70–76, 2017.

[12] F. Codevilla, M. Müller, A. López, V. Koltun, and A. Dosovitskiy, “End-to-end driving via conditional imitation learning,” in *Proc. Int. Conf. Robot. Autom.*, 2018, pp. 1–9.

[13] L. Pinto, J. Davidson, R. Sukthankar, and A. Gupta, “Robust adversarial reinforcement learning,” in *Proc. 34th Int. Conf. Mach. Learn.*, vol. 70, 2017, pp. 2817–2826.

[14] X. Pan, D. Seita, Y. Gao, and J. Canny, “Risk averse robust adversarial reinforcement learning,” in *Proc. Int. Conf. Robot. Autom.*, 2019, pp. 8522–8528.

[15] J. Hu and M. P. Wellman, “Multigent reinforcement learning: Theoretical framework and an algorithm,” in *Proc. ICML*, vol. 98, 1998, pp. 242–250.

[16] B. Baker et al., “Emergent tool use from multi-agent autocurricula,” 2019, arXiv:1909.07528.

[17] A. Dosovitskiy, G. Ros, F. Codevilla, A. López, and V. Koltun, “CARLA: An open urban driving simulator,” in *Proc. Ist Annu. Conf. Robot. Learn.*, 2017, pp. 1–16.

[18] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanciot, and N. Freitas, “Dueling network architectures for deep reinforcement learning,” in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 1995–2003.

[19] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, “Prioritized experience replay,” 2015, arXiv:1511.05952.

[20] I. Srinivas, S. Sharma, and B. Ravindran, “Dynamic frame skip deep Q-network,” 2016, arXiv:1605.03465.

[21] K. Pertsch, Y. Lee, and J. Lim, “Accelerating reinforcement learning with learned skill priorities,” in *Proc. Conf. Robot. Learn.*, 2021, pp. 188–204.

[22] K. He, G. Gkioxari, P. Dollár, and R. Girshick, “Mask R-CNN,” in *Proc. Int. Conf. Comput. Vis.*, 2017, pp. 2961–2969.

[23] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.

[24] H. Van Hasselt, A. Guez, and D. Silver, “Deep reinforcement learning with double Q-learning,” in *Proc. 30th AAAI Conf. Artif. Intell.*, 2016, pp. 2094–2100.

[25] G. Pappas, J. E. Siegel, K. Politiopolous, and Y. Sun, “A gamified simulator and physical platform for self-driving algorithm training and validation,” *Electronics*, vol. 10, no. 9, p. 1112, 2021.