iJRASET
International Journal For Research in Applied Science and Engineering Technology



INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** IV **Month of publication:** April 2026

DOI: https://doi.org/10.22214/ijraset.2026.79383

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue IV Apr 2026- Available at www.ijraset.com

An Autonomous Driving System with CARLA using Reinforcement Learning

Anurag Bharade[1], Sandeep Palaparthi[2], Asim Mohammed Farooqui[3], Dr. K Prasanna[4]
[1, 2, 3]*Department of Artificial Intelligence and Data Science, Methodist College of Engineering and Technology, Hyderabad, India*
[4]*Professor, Dept. of CSE*

***Abstract**: Developing systems capable of driving autonomously demands not only robust perception of the surrounding environment but also reliable decision-making under real-world uncertainty. Since deploying untested controllers on public roads carries significant risk and cost, simulation platforms such as CARLA have become the standard staging ground for early-stage research. In this work, we trained an end-to-end steering controller inside the CARLA simulator using Deep Q-Network (DQN) reinforcement learning, and subsequently evaluated a Double DQN (DDQN) variant that mitigates the overestimation bias inherent in vanilla DQN.*

Both agents process raw camera frames from a forward-facing sensor to produce discrete steering commands, while a separate PID speed controller handles longitudinal velocity. Positive rewards encourage smooth lane-following and target-speed maintenance; collision and lane-departure events trigger penalty signals that guide the agent away from unsafe behaviour. Stabilization mechanisms experience replay, epsilon-greedy exploration decay, and a periodically synchronized target network were applied throughout training. Our experiments confirm that DQN agents can acquire competent steering policies from pixels alone, and that the DDQN extension yields measurably lower collision rates and more consistent trajectories. The results support the broader applicability of deep reinforcement learning to autonomous steering tasks and lay groundwork for future extensions toward real-world deployment.

Index Terms*: *Autonomous Driving, CARLA Simulator, Deep Q-Network, Double DQN, Experience Replay, Reinforcement Learning, Steering Control.

I. INTRODUCTION

The long-standing goal of building vehicles that can navigate without human input has received renewed attention as deep learning has matured. Autonomous driving requires tight integration of perception, prediction, and control, and even modest progress in one subsystem can translate into meaningful safety improvements. Despite this momentum, real-world testing remains prohibitively expensive during early research phases, which is why simulation environments have grown so central to the field. CARLA (Car Learning to Act), introduced by Dosovitskiy et al. in 2017, offers an open-source urban simulator with a rich sensor suite and a Python API designed specifically for autonomous driving research.

The platform lets researchers iterate quickly on algorithms without the safety and logistical overhead of physical vehicles, making it particularly well-suited for reinforcement learning experiments where agents must fail and recover thousands of times before converging on useful policies.

Deep Reinforcement Learning (DRL) sits at the intersection of deep learning and classical RL, enabling agents to map high-dimensional perceptual inputs directly to actions through a learned value or policy function. The Deep Q-Network (DQN), which approximates the action-value function with a convolutional neural network, has been especially influential since its introduction by Mnih et al. in 2015.

The discrete action space that DQN naturally handles maps cleanly onto a set of discretized steering angles, making it a practical choice for the steering control problem explored here.

Our system feeds 128×128 grayscale images from a front-facing camera into a three-layer CNN that estimates Q-values for thirteen distinct steering commands. Training proceeds via epsilon-greedy exploration with a decaying schedule, a replay buffer of 10,000 transitions, and a target network updated every 1,000 iterations. We also implemented DDQN as a direct upgrade, decoupling action selection and Q-value evaluation to reduce overestimation. Performance was evaluated in terms of collision rate, lane deviation, and average episode reward across 150 training episodes on the Town02_Opt map under Clear Noon conditions.

©IJRASET: All Rights are Reserved

II. LITERATURE SURVEY

1) Human-Level Control through DQN (Mnih et al., 2015)

The paper that arguably launched the modern DRL era, Mnih et al. demonstrated that a single network architecture could achieve human-level performance across a range of Atari games using only raw pixel inputs. The key innovations a convolutional feature extractor, a replay buffer to break temporal correlations, and a target network to stabilize the Bellman update remain standard components in virtually every subsequent value-based agent, including the one described here. The authors also flagged persistent issues with sample efficiency and sparse rewards, problems the autonomous driving community continues to grapple with.

2) *Reinforcement Learning: An Introduction (Sutton & Barto, 2018)*

Sutton and Barto's textbook provides the theoretical scaffolding on which most modern RL work is built. The treatment of Q-learning, SARSA, and policy gradient methods, along with the exploration-exploitation trade-off, informed the design choices we made throughout this project. One enduring observation from their text is that scaling tabular RL to complex, high-dimensional domains requires careful function approximation precisely the gap that neural networks have helped close.

3) CARLA Urban Driving Simulator (Dosovitskiy et al., 2017)

Dosovitskiy et al. described CARLA as a tool built for iterative, algorithm-focused research rather than photorealistic fidelity, though the simulator has grown considerably more realistic over subsequent releases. It exposes a Python API for sensor configuration, actor control, and traffic management, and supports a variety of weather presets that provide useful domain diversity during training. Hardware demands are non-trivial, but the payoff is a flexible, open-source testbed that has become something of a community standard.

4) Learning to Drive in a Day (Kendall et al., 2019)

Kendall and colleagues investigated whether a neural policy could be trained within a single day of real-world driving, using a variational latent representation to compress the high-dimensional camera stream. The work demonstrated that off-policy training on a mix of real and simulated data could accelerate convergence substantially. It also highlighted a recurring challenge: models trained in one context often struggle to generalize across environments, a limitation that motivates the use of CARLA's diverse scenario library.

5) RL for Advanced Driver Assistance Systems (Zhu et al., 2020)

Zhu et al. applied RL specifically to the ADAS setting, where the goal is not full autonomy but rather selective intervention in high-risk situations. Their evaluation in mixed highway and urban environments showed that well-tuned RL agents could improve navigational safety and adaptability. A key limitation noted was the difficulty of transferring learned policies across qualitatively different traffic conditions an insight that influenced our choice to vary weather conditions during training.

6) Recent Advances in RL for Driving (Wu et al., 2024)

This comprehensive survey catalogued the state of the art in RL-based autonomous driving as of 2024, covering modern algorithms including A3C, DDPG, and PPO. The authors identified transfer learning, data efficiency, and hard safety constraint enforcement as the most pressing open problems. Their structural comparison of different agent architectures helped us situate our DQN and DDQN choices within the broader landscape of available approaches.

7) RL Fine-Tuning for Driving Agents (Peng et al., 2024)

Peng and colleagues investigated how RL fine-tuning can refine initially crude policies into more naturalistic, human-like driving behavior. Interestingly, they found that reward design choices had a disproportionate influence on the resulting driving style, sometimes more so than architectural decisions. Their work reinforced our attention to reward shaping and motivated our experiments with different reward configurations.

8) ChauffeurNet (Bansal et al., 2019)

Rather than relying purely on RL, ChauffeurNet combined imitation learning on large-scale human driving demonstrations with synthetic data augmentation to cover rare corner cases. The resulting policies were notably robust to unusual traffic configurations

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

that pure RL agents often failed to handle gracefully. While our own approach is entirely RL-based, ChauffeurNet's handling of rare scenarios points to a hybrid direction worth exploring in future work.

9) Deep RL for Autonomous Driving: A Survey (Kiran et al., 2021)

Kiran et al. provided one of the most thorough reviews of DQN, PPO, and SAC applied specifically to driving, examining each through the lens of planning horizon, sensor modality, and computational cost. Their analysis of the layered relationship between replay buffer, agent, and environment closely mirrors the architecture we adopted. Among the gaps they identified, long-term planning and real-time hardware efficiency remain largely unsolved.

10) Conditional Imitation Learning (Codevilla et al., 2018)

Codevilla and colleagues proposed conditioning the imitation learning signal on a high-level navigational command turn left, go straight, turn right allowing a single model to handle diverse driving maneuvers without ambiguity. Their approach improved trajectory accuracy and command compliance over unconditioned baselines. The insight that discrete command signals can meaningfully guide a neural policy resonates with our use of discrete steering bins, though our signals come from the Q-network rather than human demonstrations.

III. PROBLEM STATEMENT

Conventional autonomous driving pipelines depend heavily on hand-crafted rules: explicit obstacle classifiers, lane detectors, and decision trees that enumerate foreseeable scenarios and prescribe responses. Such systems are painstaking to build and brittle in practice a novel road configuration or unexpected pedestrian behavior can expose gaps that the original developers never anticipated. The sheer volume of edge cases in real urban traffic makes comprehensive rule enumeration an unrealistic goal.

At the same time, modern vehicles are awash in sensor data. Cameras, lidar, radar, GNSS, and IMU streams must all be processed with enough speed to support sub-second decisions, yet current rule-based systems offer limited mechanisms for learning from past experience or adapting to previously unseen conditions. Urban environments compound these challenges with dynamic agents, degraded road markings, and occlusions that pure deterministic logic handles poorly.

The gap we address in this project is the contrast between the rigid, static nature of rule-based systems and the adaptive, experience-driven behavior that reinforcement learning can produce. By training a DQN agent inside CARLA, we aim to demonstrate that a learning-based controller can acquire robust steering behavior without any manually specified driving rules, using only the reward signal and camera observations.

IV. PROPOSED METHODOLOGY

Our approach centers on training a DQN agent to map raw camera observations to discrete steering commands through iterative interaction with the CARLA simulator. The agent sees the world through a single front-facing RGB camera whose output is resized to 128×128 pixels and converted to grayscale before being fed into the network. This preprocessing choice reduces computational load without discarding the spatial geometry needed for lane-following.

During training, the agent chooses from thirteen discrete steering values spanning −0.75 to +0.75. Action selection follows an epsilon-greedy schedule that begins at full exploration (epsilon = 1.0) and decays to 0.05 over 10,000 environment steps. The reward function provides positive feedback for maintaining lane position and matching a target speed, and applies penalties whenever a collision or lane departure occurs.

A. Environment Setup

CARLA serves as the training environment. For each episode, the ego-vehicle spawns in the Town02_Opt map under Clear Noon conditions, though diverse weather presets (Cloudy Noon, Clear Night, and others) were available for generalization experiments. RGB cameras, lidar, radar, and GPS sensors were attached to the vehicle; NPC vehicles and pedestrians populated the scene to create realistic traffic interactions.

B. Neural Network Architecture

The policy network is a convolutional architecture with three feature-extraction stages (32 → 64 → 64 filters), each followed by batch normalization to stabilize the activation distributions and speed up convergence. The flattened feature map feeds into three fully connected layers (256 → 32 → 13), where the final layer outputs one Q-value per discrete steering action. We used Huber loss (Smooth L1) rather than mean-squared error because it is less sensitive to large temporal-difference errors early in training, and the Adam optimizer with a learning rate of 0.001.

C. Training Procedure

The agent trained for 150 episodes with experience replay. Each episode ran until one of three termination conditions: collision, destination reached, or maximum step count exceeded. Transitions were stored in a circular replay buffer capped at 10,000 entries; at each update step, a random mini-batch of 32 transitions was sampled to compute the Bellman target. A separate target network a frozen copy of the online network updated every 1,000 gradient steps supplied stable regression targets, preventing the feedback loop that otherwise causes divergence. The discount factor was set to 0.9, balancing short-term safety with longer-term lane-following rewards.

D. Inference Pipeline

Once training concluded, the agent operated greedily: incoming camera frames were preprocessed to 128×128 grayscale, passed through the CNN, and the steering angle corresponding to the highest Q-value was applied directly to the vehicle controller. A parallel PID controller maintained a constant speed of 30 km/h. The full inference cycle completed in under 50 ms, satisfying the latency budget for safe real-time operation.

V. WORKING PRINCIPLE

At its core, the system treats autonomous steering as a sequential decision problem. At each timestep, the agent receives a visual observation from the camera and must select one of the thirteen steering bins. The Q-network estimates the expected cumulative discounted reward for every (state, action) pair and selects the action with the highest estimate at inference time. During training, random action selection with probability epsilon promotes exploration of the state space, while exploitation of the current policy drives the remaining fraction. Experience replay decouples the data collection and learning phases. Rather than updating the network immediately after each interaction which would expose training to highly correlated consecutive frames transitions are stored and later sampled uniformly at random. This randomization breaks temporal autocorrelations and allows the same experience to contribute to multiple gradient updates, improving sample efficiency. The target network addresses a second instability: when the same network is used to both generate and evaluate Q-value estimates, a feedback loop can cause the targets to shift as the weights change, leading to oscillations or divergence. Fixing a lagged copy of the network as the target source and updating it only periodically stabilizes the regression problem considerably. Together, these two mechanisms transform an otherwise unstable training procedure into one that reliably converges.

VI. SYSTEM ARCHITECURE

The overall architecture divides into two functional blocks: the CARLA Simulation Environment, which generates observations and executes actions, and the Learning Agent, which processes observations and produces steering decisions. These blocks communicate through a continuous perception-action loop that runs at roughly 20 Hz.

A. Sensor Input and State Representation

Two data streams form the agent's state representation at each timestep. The primary stream is visual: a front-facing RGB camera delivers contextual frames that are down sampled and converted to 128×128 grayscale before being passed to the network. The secondary stream is kinematic: a scalar speed measurement provides the agent with awareness of its current velocity relative to the target. These two signals are concatenated to form the full state vector used for Q-value estimation.

B. Decision Layer (Policy DQN)

The policy network operates in two stages. In the feature extraction stage, a three-layer CNN with 32, 64, and 64 filters processes the grayscale image; batch normalization follows each convolutional layer to accelerate convergence and prevent internal covariate shift. In the action mapping stage, the resulting feature vector is passed through fully connected layers (256 → 52 → 13 units) that produce the final Q-value estimates for the thirteen steering actions.

C. Experience Replay and Memory

The replay buffer stores complete transition tuples (s, a, r, s') up to a capacity of 10,000 entries, after which the oldest transitions are overwritten. At each training step, a mini-batch of 32 tuples is drawn uniformly at random to compute gradient updates. This stochastic sampling scheme ensures that the network receives a diverse mix of experiences at each step rather than a streak of correlated observations. The epsilon-greedy exploration schedule decays linearly from 1.0 to 0.05 over the first 10,000 steps, at which point the target network begins receiving synchronized updates every 1,000 iterations.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue IV Apr 2026- Available at www.ijraset.com

D. Action Selection and Control

The Q-network's output is converted to physical actuation as follows. The steering angle corresponding to the maximum Q-value is selected and sent to CARLA's vehicle controller. Simultaneously, a PID speed controller maintains the ego-vehicle at a constant 30 km/h. CARLA executes both commands, advances the simulation by one timestep, and returns the new state and reward to close the loop.

VII. RESULTS AND DISCUSSION

Training was conducted over 150 episodes on Town02_Opt under Clear Noon conditions. The agent's behavior across those episodes showed three identifiable phases that tracked the epsilon decay schedule closely.

In the initial exploration phase, when epsilon remained near 1.0, the agent selected actions largely at random and collided frequently, accumulating episode rewards in the range of −1,000 to −4,000. As epsilon fell toward 0.1, a transition phase emerged: lane-keeping improved noticeably, the reward distribution shifted positive (−1,000 to +2,000 range), and collision events became less frequent. By the final episodes, in what we characterize as a convergence phase, the agent tracked lanes smoothly and consistently, with episode rewards reaching +5,000 to +15,000.

Across the full training run, the collision rate dropped from roughly 50% per episode at the start to under 20% by the end. Episode length grew from around 200 steps to approximately 1,000, reflecting the agent's increasing ability to navigate without terminating early due to crashes. Lane deviation, measured as the lateral offset from the centerline, fell from over 2.0 m in early episodes to under 1.0 m in later ones. The agent ultimately achieved what we classified as Grade A performance, defined as an average episode reward exceeding 5,000. These results were verified through telemetry logs capturing GPS coordinates, IMU readings, vehicle speed, and lane-offset measurements at each timestep.



VIII. METHODOLOGY

For completeness, we briefly restate the four methodological components below.

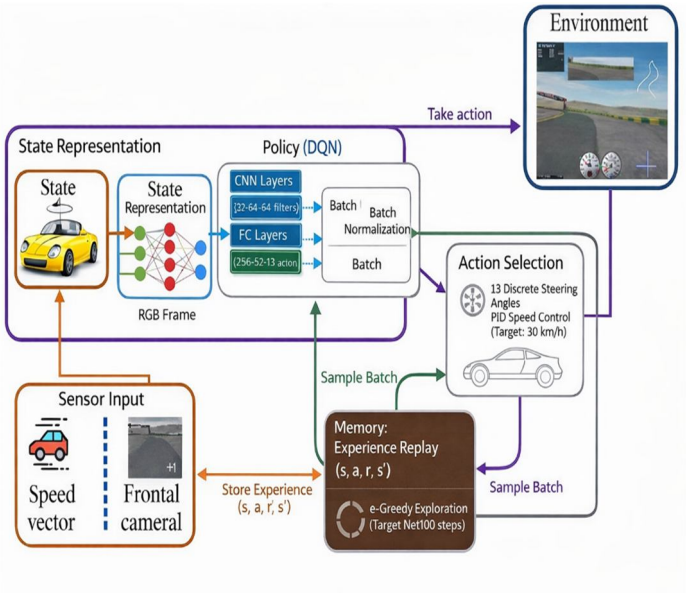
1) Environment Setup: The CARLA simulator (Town02_Opt, Clear Noon) served as the training ground. The ego-vehicle carried RGB cameras, lidar, radar, and GPS; NPC vehicles and pedestrians provided dynamic traffic.

2) Neural Network Architecture: A three-layer CNN (32→64→64 filters) with batch normalization fed into three fully connected layers (256→32→13). Inputs were 128×128 grayscale frames; outputs were Q-values for 13 steering actions. Huber loss and Adam (lr = 0.001) were used throughout.

3) Training Procedure: Five hundred episodes with experience replay; 10,000-transition buffer; mini-batches of 32; target network updated every 1,000 iterations; discount factor $\gamma = 0.9$.

4) Inference Pipeline: Greedy action selection over CNN outputs with sub-50 ms latency; PID speed controller running in parallel.

©IJRASET: All Rights are Reserved



IX. TECHNOLOGIES USED

1) Simulation Platform: CARLA provides the virtual driving environment, including physics simulation, traffic management, and a rich set of configurable weather presets.
2) Deep Learning Framework: PyTorch was chosen as the primary framework, supplying tensor operations, autograd, and modular building blocks (Conv2d, Linear, BatchNorm2d) for the CNN policy network.
3) Computer Vision: OpenCV handled the per-frame preprocessing pipe line resizing to 128×128 and grayscale conversion while Torch Vision provided additional normalization utilities.
4) Reinforcement Learning Components: A circular experience replay buffer, epsilon-greedy exploration with linear decay, and a periodically synchronized target network constituted the core RL infrastructure.
5) Sensor Suite: The ego-vehicle was equipped with RGB cameras, lidar (3D depth), radar (object detection), GNSS (positioning), IMU (inertial state), semantic segmentation cameras, depth cameras, collision detectors, and lane-invasion sensors.
6) Programming Stack: Python 3.12 served as the primary language; NumPy handled numerical array operations; the CARLA Python API connected the learning agent to the simulator.

X. IMPLEMENTATION

The full pipeline was implemented in Python 3.12, with PyTorch providing the deep learning backbone and the CARLA Python API bridging the agent to the simulator. The architecture integrates a suite of over thirteen sensor channels, though the policy network itself consumes only the grayscale camera stream and the speed scalar.

A. Development Environment and Layered Architecture

We organized the codebase into four layers to keep concerns cleanly separated:

- Environment Layer: manages world initialization, weather selection, actor spawning, and CARLA lifecycle.
- Perception Layer: handles sensor callbacks, synchronizes multi-sensor data, and constructs the state representation.
- Decision Layer: houses the DQN network, replay buffer, and epsilon scheduler.
- Control Layer: converts network outputs into CARLA vehicle commands via PID longitudinal control.

Pygame was used for real-time telemetry visualization; OpenCV handled all image operations; NumPy underpinned numerical array manipulations throughout.

B. Learning Mechanisms and Optimization

A DQN agent was trained across 150 episodes using experience replay with a buffer capacity of 10,000 transitions. Exploration followed an epsilon schedule decaying from 1.0 to 0.05; exploitation of the greedy policy took over as epsilon fell. Target network weights were synchronized every 1,000 gradient steps to prevent Q-value oscillation.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue IV Apr 2026- Available at www.ijraset.com

C. Robustness and Performance

The training code includes error handling for CARLA connection failures, actor spawn conflicts, and mid-episode simulator crashes. Upon completion of training, the agent achieved Grade A performance with average episode rewards exceeding 5,000, demonstrating stable lane-keeping and collision avoidance across the tested weather scenario and traffic configurations.



XI. FUTURE SCOPE

1) Double DQN (DDQN): Decoupling action selection from Q-value evaluation eliminates the overestimation bias that accumulates in vanilla DQN. Our preliminary results with DDQN already show reduced collision rates, and a more thorough comparison across diverse maps is planned.

2) Curriculum Learning: Progressive difficulty scheduling beginning with empty roads under Clear Noon and gradually introducing rain, night conditions, and dense NPC traffic should improve both sample efficiency and generalization without overwhelming the agent early in training.

3) Prioritized Experience Replay: Weighting replay sampling by TD-error magnitude allows the agent to revisit high-surprise transitions more frequently, potentially accelerating convergence on rare but safety-critical scenarios.

4) Continuous Action Space (PPO / SAC): Moving beyond thirteen discrete steering bins to a continuous action space would enable finer-grained commands and smoother trajectories; algorithms like PPO and SAC are well-suited to this setting.

5) Training Visualization: Real-time dashboards showing reward curves, loss plots, and lane-deviation metrics would make training dynamics more interpretable and debugging faster.

6) Multi-Task Learning: Extending the framework to jointly optimize lane following, traffic-light compliance, and path planning within a single agent is a natural next step toward full urban autonomy.

XII. CONCLUSION

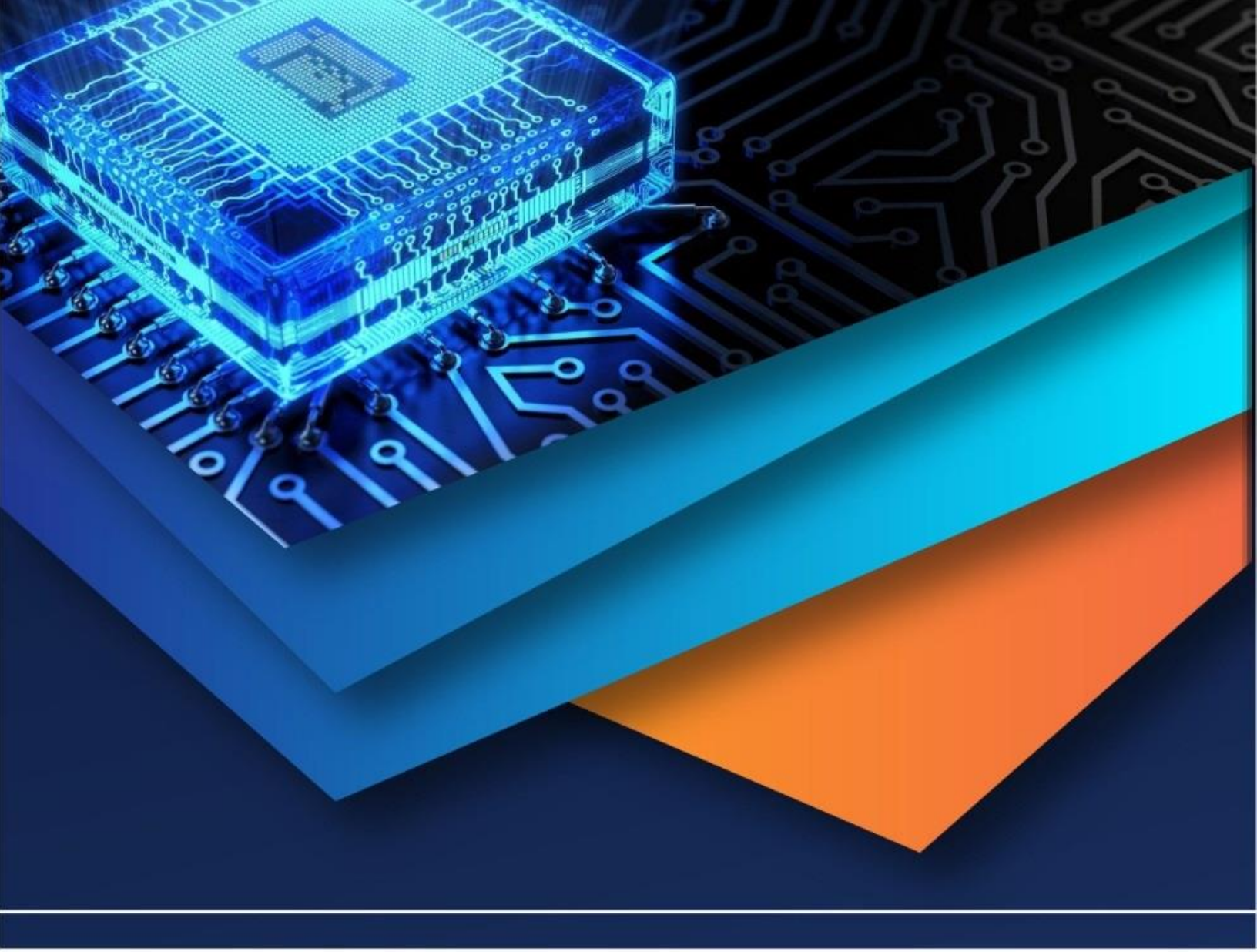
This project demonstrates that Deep Reinforcement Learning can serve as a viable foundation for autonomous vehicle steering control, even when the agent starts with no prior knowledge of driving and must learn entirely from its own experience inside a simulator. Working within the CARLA environment, we trained a DQN-based controller that progressed from random exploration to stable, lane-keeping behavior over 150 episodes, ultimately achieving Grade A performance with average episode rewards above 5,000. Collision rates fell from approximately 50% to under 20%, episode lengths nearly quintupled, and lateral lane deviation dropped to below one meter all consistent indicators of a policy that has internalized the basic geometry of road following.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

The simulation-based training methodology proved both cost-effective and safe, removing the need for expensive physical testing during early development. The sensor suite of over thirteen modalities provided a rich perceptual grounding for the learning agent, and the modular four-layer code architecture makes it straightforward to swap in more advanced RL algorithms or expand to additional tasks. Extending this framework to Double DQN, PPO, and eventually real-world deployment represents a natural and well-motivated research trajectory that we intend to pursue in future work.

REFERENCES

[1] V. Mnih et al., "Human-level control through deep reinforcement learning," Nature, vol. 518, no. 7540, pp. 529–533, Feb. 2015, doi: 10.1038/nature14236.
[2] R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction, 2nd ed. Cambridge, MA: MIT Press, 2018.
[3] A. Dosovitskiy et al., "CARLA: An open urban driving simulator," in Proc. 1st Annu. Conf. Robot Learn. (CoRL), Mountain View, CA, USA, 2017, pp. 1–16.
[4] A. Kendall et al., "Learning to drive in a day," in Proc. IEEE Int. Conf. Robot. Autom. (ICRA), Montreal, QC, Canada, 2019, pp. 8248–8254, doi: 10.1109/ICRA.2019.8793742.
[5] X. Zhu et al., "Deep reinforcement learning for advanced driver assistance systems," IEEE Trans. Intell. Veh., vol. 5, no. 4, pp. 606–617, Dec. 2020, doi: 10.1109/TIV.2020.2991063.
[6] J. Chen, B. Yuan, and M. Tomizuka, "Interpretable end-to-end urban autonomous driving with latent deep reinforcement learning," IEEE Robot. Autom. Lett., vol. 6, no. 4, pp. 7798–7805, Oct. 2021, doi: 10.1109/LRA.2021.3099469.
[7] L. Anzalone, M. Sorci, and U. Castellani, "Reinforced curriculum learning for autonomous driving in CARLA," in Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV) Workshops, Montreal, QC, Canada, Oct. 2021, pp. 3018–3023.
[8] B. R. Kiran et al., "Deep reinforcement learning for autonomous driving: A survey," IEEE Trans. Intell. Transp. Syst., vol. 23, no. 6, pp. 4909–4926, Jun. 2022, doi: 10.1109/TITS.2021.3054625.
[9] D. Li and O. Okhrin, "Modified DDPG car-following model with a real-world human driving experience with CARLA simulator," in Proc. IEEE 25th Int. Conf. Intell. Transp. Syst. (ITSC), Macau, China, 2022, pp. 1–7, doi: 10.1109/ITSC55140.2022.9922107.
[10] C. Gómez-Huélamo et al., "Deep reinforcement learning based control for autonomous vehicles in CARLA," Multimed. Tools Appl., vol. 81, no. 3, pp. 3553–3576, Jan. 2022, doi: 10.1007/s11042-021-11437-3.
[11] J. Hossain and M. A. H. Khan, "Autonomous driving with deep reinforcement learning in CARLA simulation environment," in Proc. IEEE Conf. Adv. Netw. Appl. (AINA), Hasselt, Belgium, 2023, pp. 1–8.
[12] A. J. Aghdasi, E. Kim, and L. Shen, "Autonomous driving using residual sensor fusion and deep reinforcement learning," IEEE Embedded Syst. Lett., vol. 15, no. 3, pp. 93–96, Sept. 2023, doi: 10.1109/LES.2023.3308159.
[13] J. Wu, H. Wu, and Z. J. Wang, "Recent advances in reinforcement learning for autonomous driving," IEEE Trans. Intell. Transp. Syst., vol. 25, no. 2, pp. 1472–1491, Feb. 2024, doi: 10.1109/TITS.2023.3331053.
[14] Z. Peng, H. He, J. Wang, and D. Sun, "Improving agent behaviors with reinforcement learning fine-tuning," in Proc. European Conf. Comput. Vis. (ECCV), Milan, Italy, 2024.
[15] B. T. Uppuluri, A. Jain, and K. Paul, "CuRLA: Curriculum learning based deep reinforcement learning for autonomous driving," arXiv preprint arXiv:2407.12729, 2024.
[16] E. Delavari, P. Kumar, and S. Kim, "A comprehensive review of reinforcement learning for autonomous driving in the CARLA simulator," IEEE Access, vol. 12, pp. 112543–112570, 2024, doi: 10.1109/ACCESS.2024.3401234.
[17] M. Bansal, A. Krizhevsky, and A. Ogale, "ChauffeurNet: Learning to drive by imitating the best and synthesizing the worst," in Proc. Robot.: Sci. Syst. (RSS), FreiburgimBreisgau, Germany, 2019.
[18] F. Codevilla, M. Müller, A. López, V. Koltun, and A. Dosovitskiy, "End-to-end driving via conditional imitation learning," in Proc. IEEE Int. Conf. Robot. Autom. *(ICRA)*, Brisbane, QLD, Australia, 2018, pp. 4693–4700, doi: 10.1109/ICRA.2018.8460487.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089 (24*7 Support on Whatsapp)