Feasibility of Deep Reinforcement Learning on Edge For Autonomous Driving

ANONYMOUS AUTHOR(S)

Autonomous Driving System (ADS) integrates AI, sensors, and communication networks to enhance safety, traffic efficiency, and sustainability. The adoption of machine learning (ML) techniques provides a better handling of the complexities and unpredictability of real-world scenarios. ML methods such as Deep Reinforcement Learning (DRL) leverage deep learning (DL) for robust feature extraction combined with reinforcement learning's sequential decision-making capabilities in dynamic environments. However, DRL often demands significant computational power—typically supported by cloud servers or GPUs and communication between edge based ADS and cloud based intelligence may lead to challenges such as high latency, reduced reliability, and elevated costs. To address this, we explore deploying the DRL model integrated with autoencoder onto a low-powered edge device. This work especially focuses on the Proximal Policy Optimization (PPO) of the DRL model and improves them with trainable exploration rate and Generalized Advantage Estimations to handle high dimensional space. This approach is evaluated using a Processor-in-the-Loop (PIL) framework integrated with the realistic 3D CARLA simulation to achieve a coherent real-time interaction loop between the virtual environment and the edge. The edge prediction for this process achieved an inference time of 36.85 ms with a Root Mean Square Error (RMSE) of 0.0086. To the best of our knowledge, this is the first effort to show the practical feasibility and effectiveness of DRL model deployment on edge devices for real-time autonomous driving applications.

Additional Key Words and Phrases: DRL, Edge Prediction, ADS, PIL

ACM Reference Format:
Anonymous Author(s). 2025. Feasibility of Deep Reinforcement Learning on Edge For Autonomous Driving. 1, 1 (March 2025), 17 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

An Automotive Driving System (ADS) refers to the integration of advanced technologies, such as sensors, AI, communication networks, etc., to enable vehicles to navigate and operate autonomously or with minimal human intervention. It will enhance safety by reducing human error and improve traffic efficiency [15]. Moreover, it contributes to sustainability by optimizing fuel consumption and enabling smarter urban mobility solutions[4]. Traditional ADS followed rule-based, modular approach, where specific features like adaptive cruise control, lane-keeping assist, and parking assist were designed as standalone components [20]. These systems relied on pre-programmed rules and algorithms to handle specific scenarios, such as maintaining a safe distance from other vehicles or staying within lane boundaries. For these reasons, it faced limitations due to the dynamic and unpredictable nature of real-world environments, such as diverse road conditions, weather variations, unexpected obstacles, etc. Rule-based systems struggled to adapt to these complexities, leading to a stagnation in the progression toward full autonomy[20]. To overcome these challenges, modern ADS are designed with AI and machine learning (ML) techniques [10].

ACM XXXX-XXXX/2025/3-ART
https://doi.org/10.1145/nnnnnnn.nnnnnnn

The integration of ML methods into the automobile domain spans a wide spectrum, ranging from simpler control tasks, such as braking, to highly complex operations like fully autonomous driving [1]. Techniques applied in this field vary significantly, encompassing supervised learning approaches like Deep Neural Networks (DNN) as well as more complex methodologies such as Reinforcement Learning (RL) [5]. However, the ML applications in ADS face many challenges, such as DNN, which is more effective for classification and feature extraction tasks but heavily depends on comprehensive datasets, which are not always readily available in the context of autonomous driving. Additionally, these methods often fail to adequately handle applications involving sequential decision-making and dynamically evolving environments [9]. In contrast, Reinforcement Learning (RL) based techniques, such as Deep Q-Networks (DQN) and Policy Gradient methods, have demonstrated superior adaptability in handling complex driving scenarios [11]. Model-based reinforcement learning is explored to mitigate the high data dependence issue by incorporating simulation-based approaches for autonomous driving training. Additionally, safety-driven learning frameworks, such as safe RL and constrained optimization techniques, are developed to ensure autonomous systems make decisions while minimizing the risk of collisions. Currently, many autonomous driving systems are increasingly adopting Deep Reinforcement Learning (DRL) due to their interactive nature and inherent reward-punishment mechanisms [11]. DRL, in particular, demonstrates effectiveness in managing dynamic and uncertain environments by enabling vehicles to learn and adapt interactively [5].

However, deploying DRL models directly on edge devices presents significant challenges, primarily due to the high computational and memory demands associated with processing raw, high-dimensional sensory inputs such as images. These constraints limit the feasibility of real-time inference and decision-making on resource-constrained hardware platforms. To address this issue, a Variational Autoencoder (VAE) [6, 16] can be integrated into the DRL pipeline to compress high-dimensional observations into a low-dimensional latent representation (commonly referred to as the z-dimension). This dimensionality reduction significantly decreases the input complexity for the policy (Actor) network, enabling lightweight and efficient inference suitable for edge deployment. The VAE architecture not only reduces computational overhead but also preserves the most salient features required for effective policy learning. Furthermore, its probabilistic encoding mechanism enhances robustness to noise and variability in sensory data, which is crucial for maintaining performance in dynamic real-world driving environments. This integrated VAE-DRL approach provides a promising solution for achieving scalable, adaptive, and efficient autonomous navigation on edge computing platforms.

1.1 Need for Simulation Software and Edge Deployment

Training and testing costs, as well as safety considerations, are being effectively managed using high-end simulation software like CARLA and TORCS [3, 26]. Nonetheless, the critical necessity for split-second decision-making and robust security introduces additional complexities, especially concerning latency and security issues when deploying these sophisticated models on cloud servers or high-end GPUs [13].

Considering these points, this is the first work that proposes a framework deploying the trained DRL models specifically tailored for autonomous driving onto resource-constrained edge devices embedded within vehicles. Moreover, our framework suggests integrating Processor-in-the-Loop (PIL) methods with high-fidelity 3D simulation software rather than relying exclusively on physical hardware testing. This strategy leverages advanced 3D simulation environments, which can accurately mimic real-world scenarios, thereby demonstrating the feasibility and effectiveness of deploying DRL models on edge platforms for autonomous driving.

1.2 The key contributions

- This is the first study which explores edge-based DRL implementation for autonomous tasks such as collision avoidance, lane-keeping, speed regulation, and optimal path planning.
- The DRL model, integrated with a VAE, is trained on CARLA 3D simulation using a GPU. A dataset is then generated from the trained model, followed by model quantization to balance computational latency and accuracy for efficient edge deployment. Finally, the quantized model is evaluated directly on edge hardware to assess computational timing and loss performance without approximations.
- The Processor-in-the-Loop (PIL) technique consolidates the GPU and edge modules into a cohesive real-time interaction loop. This facilitates the high-performance GPU-based simulation to dynamically interact with the resource-limited edge device, which is tasked with computing real-time policy decisions.

2 RELATED WORK

Numerous methods and techniques are employed in the domain of autonomous driving, ranging from traditional rule-based systems to advanced machine learning algorithms. Despite the progress made, each of these methods has limitations that must be addressed to develop truly autonomous systems capable of safely and efficiently navigating real-world environments.

Models like Convolutional Neural Networks (CNNs) are effectively utilized by NVIDIA Corporation to map raw pixels from camera inputs directly to steering commands, demonstrating a novel end-to-end learning approach in autonomous driving [1]. These requirements pose significant barriers, particularly for edge computing environments where resource constraints are a critical challenge. Hence, CNNs often need to be supplemented with more adaptive frameworks like Deep Reinforcement Learning (DRL) [17, 21, 23] to handle decision-making in complex and dynamic driving environments effectively. Reinforcement learning techniques, such as Deep Q-Networks (DQN) [21], are demonstrated remarkable efficiency in tasks like playing Atari games and robotic control. However, they face significant challenges, particularly when dealing with continuous action spaces or high-dimensional input data. In such cases, the "curse of dimensionality" [8, 24] makes it increasingly difficult to estimate Q-values accurately as the state-action space grows exponentially. Furthermore, DQN [21] can become unstable or diverge due to the complexities introduced by Bellman's equation [24] when applied to such environments. These challenges highlight the need for alternative methods like policy gradient approaches, which are better equipped to handle the intricacies of continuous and high-dimensional settings.

Recent works [5, 11, 17] in the autonomous driving domain using DRL, such as Deep Deterministic Policy Gradient (DDPG) and Proximal Policy Optimization (PPO), are achieved significant results in areas like lane deviation minimization, collision avoidance, and optimal path planning. Proximal Policy Optimization (PPO) is better suited for edge computing environments compared to Deep Deterministic Policy Gradient (DDPG)[23] due to its relative simplicity and computational efficiency. While DDPG utilizes a more complex architecture involving four neural networks (actor, target actor, critic, target critic) [23] and a replay buffer, PPO employs a more streamlined approach that eliminates the need for a replay buffer and focuses on directly optimizing the policy. This simplicity reduces memory usage and computational overhead, making PPO particularly advantageous for resource-constrained edge devices [17, 21]. However, these advancements rely heavily on expensive setups, such as GPUs, making them less feasible for deployment in resource-constrained environments [13, 22]. Notably, minimal resource setups like embedded or edge computing devices are often overlooked in these studies. These setups present unique challenges, including low computational power, limited memory, and stringent power consumption requirements [27]. Deploying

DRL models on edge requires careful selection of a model with minimal complexity to meet the constraints of limited computational resources. For this purpose, Proximal Policy Optimization (PPO) with an actor-critic architecture is a suitable choice due to its balance of efficiency and performance [17].

To further reduce computational demands, the DRL model is supplemented with a Variational Autoencoder (VAE). This integration enables the use of a low-dimensional latent space, significantly minimizing the computational load while maintaining robust decision-making capabilities [7]. Moreover, research has explored quantization techniques, such as post-training quantization and knowledge distillation, to compress deep learning models for edge AI applications. Pruning methodologies have also been employed to optimize neural network architectures, reducing the number of computations required while preserving essential task performance [14]. Additionally, attention-based mechanisms and lightweight convolutional networks have been introduced to enhance the efficiency of vision-based DRL models for real-time deployment [25]. These techniques pave the way for scalable, energy-efficient AI solutions for real-world autonomous driving applications.

3 PRELIMINARIES

This section provides a glimpse of the existing concepts used in this study. This work employs the CARLA simulation environment for training and evaluation. In the training environment, we integrate a Variational Autoencoder (VAE) to handle high-dimensional sensory data efficiently. Moreover, the control policy for autonomous driving is implemented using the Proximal Policy Optimization (PPO) algorithm, which leverages an actor-critic architecture for stable and efficient reinforcement learning. Reward shaping is employed to encourage the agent to take desirable actions and penalize undesirable ones. The detailed explanation is as follows:

3.1 CARLA Environment

CARLA (Car Learning to Act) [3] is an open-source simulator designed for autonomous driving research, developed by the Computer Vision Center (CVC). It provides a high-fidelity environment for training and testing reinforcement learning (RL) models in realistic urban scenarios [2]. It offers a flexible API for controlling vehicles, pedestrians, and infrastructure elements, making it suitable for reinforcement learning applications such as behavior prediction, decision-making, and end-to-end autonomous driving. One of the key reasons CARLA is essential for autonomous driving (AD) training is its ability to provide a controlled and cost-effective alternative to real-world testing. CARLA provides different layouts like rural (town 1) and urban (town 2) with their own road patterns and obstacle complexities.

3.2 Varaiational Auto Encoder

A Variational Autoencoder (VAE) [6, 16] consists of three main components: the encoder, the latent space z_{dim}, and the decoder. The encoder includes a Convolutional Neural Network (CNN) for feature extraction of the input data, such as lane markers, sidewalks, roads, pedestrians, etc. Afterward, the features are compressed into a lower-dimensional latent representation by mapping them to a probability distribution characterized by a mean (μ) and standard deviation (σ). This ensures that similar inputs produce similar latent vectors, enabling smooth interpolation. The latent space (z-space) serves as the probabilistic bottleneck where sampling occurs, introducing a degree of randomness that enhances the model's generalization ability. The decoder reconstructs the original data from the sampled latent variables, effectively learning meaningful representations. In our work, VAE is utilized solely for latent space representation, which are then provided to

reinforcement learning agent, significantly reducing the computational burden while preserving task-relevant information.

3.3 Actor-Critic Network

The *Actor-Critic* (AC) [12, 23] architecture is a widely used framework in RL [16] that combines policy-based and value-based methods to optimize decision-making in complex environments. The actor is responsible for selecting actions based on the current policy. It maps states to actions using a parameterized policy $\pi_\theta(a|s)$, which can be either deterministic or stochastic. The actor continuously updates its policy parameters (θ) to maximize expected rewards. The critic evaluates the actions taken by the actor by estimating the value function. It learns to approximate either the *state-value function* $V(s)$ or the *advantage function* $A(s, a)$, which helps in reducing the variance of policy updates. The critic provides feedback to the actor to guide policy improvement. The *Actor-Critic* method addresses key challenges in RL by balancing exploration and exploitation while leveraging value estimates for stable learning. This architecture forms the foundation for advanced algorithms such as *Advantage Actor-Critic (A2C)*, *Proximal Policy Optimization (PPO)*[**?**], and *Deep Deterministic Policy Gradient (DDPG)*[23], which are widely used in continuous control tasks, including autonomous driving in CARLA.

3.4 Reward Shaping

We adopt a dense reward function [16], R, which encourages our agent to meet the objectives such as collision avoidance, staying in the lane, set velocity range, and being able to reach the predefined distance.

$$R = \begin{cases} -N_o, & \text{on infraction} \\ v_{\min} \times (1 - d_{\text{norm}}) \times R_o, & v < v_{\min} \\ 1 \times (1 - d_{\text{norm}}) \times R_o, & v_{\min} \leq v < v_t \\ (1 - \frac{v - v_t}{v_{\max} - v_t}) \times (1 - d_{\text{norm}}) \times R_o, & v \geq v_t \end{cases} \tag{1}$$

In the above equation, R considers the agent's speed, its alignment with the lane, and penalizes any infractions. The distance between the vehicle's center and the lane center is critical for safe driving, with performance improving as this distance decreases. Hence, the distance is normalized as $d_{\text{norm}} = \frac{d}{d_{\max}}$, where $d_{\max}$ is a predefined threshold used in the termination condition. The orientation reward, which plays a key role in steering control, is computed as $R_o = 1 - \frac{|\alpha_{\text{diff}}|}{\alpha_{\max}}$ when $\alpha_{\text{diff}} < \alpha_{\max}$; otherwise, the reward is set to zero. The agent aims to maximize its reward by achieving the target speed, staying close to the lane center, and maintaining correct orientation while avoiding infractions.

3.5 Proximal Policy Optimization

Proximal Policy Optimization (PPO) algorithm [16, 17] is considered in this study due to its effectiveness in constraining policy updates within safe range, thereby enhancing training stability and convergence. It builds upon the foundational ideas introduced by Trust Region Policy Optimization (TRPO) [18], which limits policy changes to a trust region in order to prevent large, destabilizing updates during training. PPO achieves a similar effect but replaces TRPO's complex optimization constraints with a clipped objective function, making it significantly more computationally efficient and easier to implement in practical scenarios.

In the above algorithm:

Algorithm 1 Proximal Policy Optimization (PPO)

Initialize policy π_θ , value function parameters π_ϕ,exploration noise σ_{noise}, and buffer B
for each iteration $k = 1, \ldots, K$ **do**
  for each episode $n = 1, \ldots, N$ **do**
    Collect trajectories using old policy $\pi_{\theta_{old}}$
    Store transitions (s_t, a_t, r_t, d_t) in buffer B
  end for
  Compute advantage estimates $\hat{A}_t$ and value targets V_{target}
  for each update step $i = 1, \ldots, n_{\text{updates}}$ **do**
    Compute sampling ratio $r_t(\theta)$
    Compute total policy loss $L_{\text{policy}} = -L_{\text{clip}} - \beta L_{\text{entropy}}$ and
    value function loss L_{value}
    Compute gradients and update parameters:
    $\theta \leftarrow \theta - \eta \nabla_\theta L_{\text{policy}}$
    $\phi \leftarrow \phi - \eta \nabla_\phi L_{\text{value}}$
  end for
  Update old policy weights: $\pi_{\theta_{\text{old}}} \leftarrow \pi_\theta$
  Update old value function weights: $\pi_{\phi_{\text{old}}} \leftarrow \pi_\phi$
  Clear buffer B
end for

- The ratio $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ compares the probability of the action under the current and old policies, used to control the size of the policy update.
- The advantage function $\hat{A}_t = r_t + \gamma V(s_{t+1}) - V(s_t)$ at timestep t guides the updates in the direction of better performance, where $\gamma \in [0.95, 0.99]$ is the discount factor, typically chosen to balance short and long-term rewards in reinforcement learning.
- The value loss function $L_{\text{value}} = \frac{1}{2}\mathbb{E}_t\left[(V_\phi(s_t) - V_{\text{target}})^2\right]$ measures the mean squared error between the predicted value and the target return.
- In the policy loss function L_{policy}, the $L_{\text{clip}} = \mathbb{E}_t\left[\min\left(r_t\hat{A}_t, \text{clip}(r_t, 1-\epsilon, 1+\epsilon)\hat{A}_t\right)\right]$ is the clipped surrogate objective, which ensures stable and conservative updates by limiting large policy changes. Moreover, the $L_{\text{entropy}} = \sum_{i=1}^{d}\left(\frac{1}{2}\log(2\pi\sigma_i^2) + \frac{1}{2}\right)$ represents the entropy of the Gaussian policy distribution and encourages exploration in action selection. In L_{policy}, $\epsilon \in [0.1, 0.3]$ is the clipping threshold that controls the deviation of the new policy from the old one and $\beta \in [0.001, 0.01]$ is the entropy coefficient that balances exploration versus exploitation.
- In L_{entropy}, Standard deviation σ_i for the i^{th} action dimension is controlled by exploration noise σ_{noise} and d is the number of action dimensions.

4 PROPOSED FRAMEWORK

This work explores end-to-end prediction, as illustrated in Fig. 1, using a PPO-based RL agent (actor and critic networks) integrated with a VAE, which compresses high-dimensional semantic images into a low-dimensional latent space (z-dim). This enables efficient policy learning while reducing computational overhead. Afterwards, the VAE and the actor model are quantized and deployed onto a low-cost edge device to enable real-time inference under resource constraints. To overcome the challenges of physical hardware testing and to enable closed-loop evaluation, we implemented a Processor-in-the-loop (PIL) framework that bridges the CARLA simulation environment with edge-based computation.

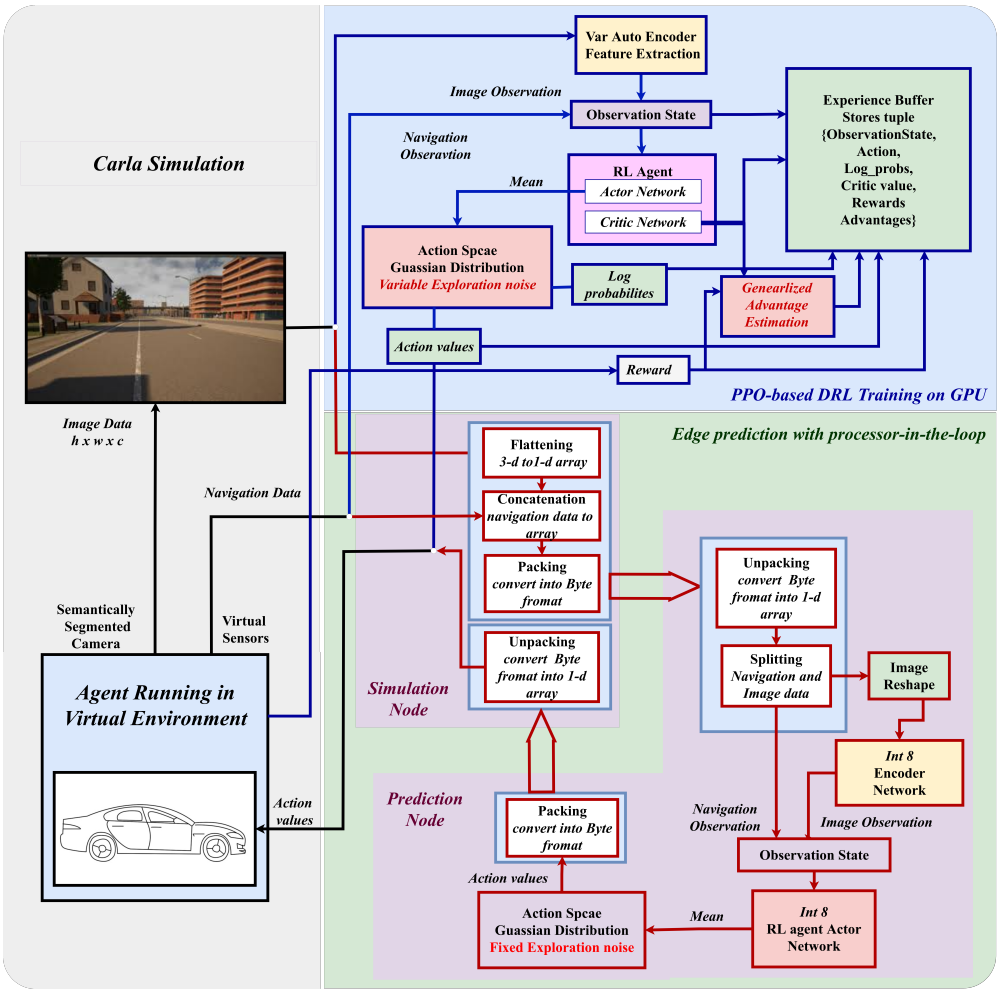


Fig. 1. End-to-End DRL-Based Prediction with PPO in a Processor-in-the-loop Framework

4.1 PPO-based DRL Training on GPU

In the Fig. 1, the training framework incorporates an agent built by pipelining the VAE and the DRL model that leverages Proximal Policy Optimization (PPO) for decision-making in the CARLA simulated environment.

4.1.1 Training Model (VAE and DRL). In the simulation, VAE is employed for feature extraction of lane markers, sidewalks, roads, pedestrians, etc. and it provides the dimensionality reduction of the feature map through the latent space, as mentioned in section 3.2. The input (160 × 80 RGB image) , specifically a semantically segmented (SS) frame) is fed to VAE, as illustrated in Fig. 2.The dataset used for training the VAE was sourced from a publicly available repository[16], where data was collected by manually driving a vehicle in the CARLA simulator to record observations and actions across diverse urban scenarios and fed to the VAE to processes the input image through a series of Convolution layers to generate a 95-dimensional latent vector, which captures the spatial and semantic structure of the scene. With this latent space additional navigation features like vehicle

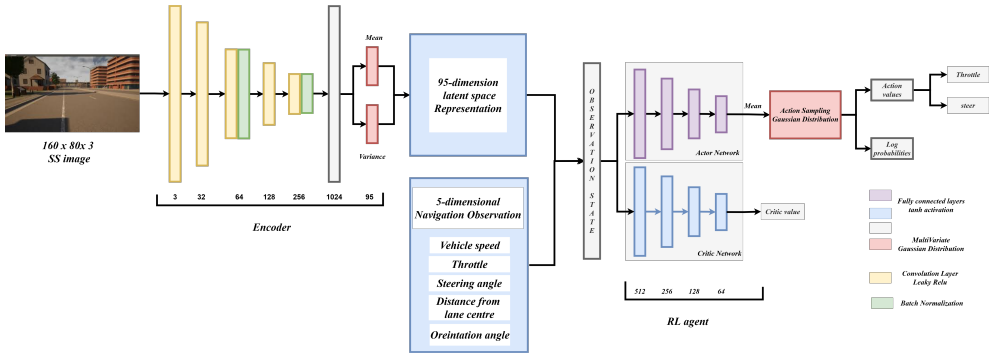


Fig. 2. Training Model (VAE integrated with RL agent)

speed, throttle, steering angle, distance from the lane center, and orientation angle are concatenated, resulting in a 100-dimensional observation state. This observation vector combines both visual and dynamic driving state information, which serves as the input to the RL agent. The RL agent (DRL model) uses the actor-critic architecture, as mentioned in section 3.3, to make sequential decisions. In this model, the actor is responsible for generating mean values based on the current state, as depicted in Fig. 2. Using these mean values an action (throttle and steer) is sampled from gaussian distribution, while the critic evaluates the chosen actions by estimating their value, guiding the learning process. It is important to note that DRL model is trained using PPO algorithm to keep policy updates within stable range, which is explained below.

4.1.2 PPO. This algorithm [16] follows the clipped surrogate function, as mentioned in section 3.5, to train the RL agent. It is essential to collect trajectories consisting of the agent's states, actions, and the corresponding rewards obtained through interaction with the environment. These collected trajectories are then used to optimize the policy using the clipped surrogate objective, enabling stable updates while leveraging the estimated advantages from the agent's experiences. For this purpose, the CARLA virtual environment, as discussed in section 3.1, is used to run the RL agent. As a virtual training environment, 'town 2' of CARLA is chosen due to its urban construct and road patterns, such as elbow turns and T-junctions, which are commonly found in real-world urban towns, as they introduce complex navigation challenges.

4.1.3 Trajectory Collection and Policy Update. The agent executes its policy $\pi_{\theta_{\text{old}}}$ in the environment for T time-steps to collect tuples of the form $(s_t, a_t, r_t, d_t, V(s_t; \theta_v))$, which are stored in experience buffer on episode basis. These trajectories include environmental feedback based on the agent's actions, incorporating factors such as safety, lane-keeping, and driving efficiency. The agent receives a reward signal derived from these factors, with the reward function guiding the learning process by reinforcing desirable actions and discouraging undesirable ones. Each episode follows a predefined route in the Town 2 environment, and successful completion is marked by the agent reaching its destination. Episodes are terminated upon meeting predefined termination conditions such as collisions, deviation from the lane, or exceeding the maximum allowed episode length. Agent is run for N such episodes gathering the data to form a dataset for training and and used to compute the Policy and the value loss (as described in Section 3.5) to update both the actor and critic networks by calculating gradients. The new policy is updated using the clipped surrogate objective, ensuring that it does not deviate significantly from the old policy and thereby maintaining stable training.

4.1.4 Variable Exploration Noise (σ_{noise}). In PPO-based policy, exploration is introduced by sampling actions from a Gaussian distribution parameterized by the predicted mean $\mu_\theta(s_t)$ and standard deviation σ_{i}. The exploration noise which is defined by a hyperparameter σ_{noise} acts as standard deviation, allowing the policy to sample varied actions around the mean rather than producing deterministic outputs. The action distribution in our Proximal Policy Optimization (PPO) implementation is defined as:

$$\Sigma_\theta = \text{diag}(\sigma_{\text{i}}) \tag{2}$$

where $\mu_\theta(s_t)$ represents the mean of the action distribution, and Σ_θ is a diagonal covariance matrix constructed directly from the exploration standard deviation σ_θ, such that:

$$a_t \sim \pi_\theta(a_t \mid s_t) = \mathcal{N}(\mu_\theta(s_t), \Sigma_\theta) \tag{3}$$

To enhance exploration efficiency, we adopt a variable exploration noise strategy by modifying the standard deviation σ_{noise} over time. Unlike fixed noise levels, a time-varying standard deviation allows the agent to dynamically adjust its exploration behavior as training progresses. σ_{i} is assigned based on the current value of σ_{noise}, which is varied across training phases. Initially, a higher value of σ_{noise} encourages broad exploration by introducing more randomness into the actions. As training progresses and the policy improves, σ_{noise} is gradually reduced, resulting in more deterministic and refined actions that reflect the learned behavior of the agent.

4.1.5 Generalized Advantage Estimation (GAE). In autonomous driving tasks, to improve the accuracy of advantage calculations during policy training, where feedback signals can be sparse or delayed and decisions must be made in rapidly changing environments, GAE [19] helps reduce the variance of policy gradient estimates while preserving bias within acceptable limits. By leveraging a weighted sum of temporal differences across multiple time steps, GAE captures the long-term impact of actions more effectively than simple advantage estimates.

The advantage function using GAE is computed as:

$$\hat{A}_t = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l} \tag{4}$$

where δ_t is the temporal difference (TD) residual given by:

$$\delta_t = r_t + \gamma V_\phi(s_{t+1}) - V_\phi(s_t) \tag{5}$$

The parameter $\lambda \in [0, 1]$ controls the bias-variance tradeoff, where higher λ values reduce bias but increase variance. By integrating GAE, our PPO implementation ensures more stable training, reducing high variance in gradient estimates while retaining sufficient exploration for improved policy learning.

4.2 Edge prediction with Processor-in-the-loop

The Processor-in-the-loop (PIL) framework, illustrated in Fig. 1, integrates both the server (simulation node) and the edge nodes in a coherent real-time interaction loop. An ethernet-based communication is established to interconnect these two nodes. Each node performs remote procedure calls (RPCs) to enable seamless bidirectional data exchange in real-time. The simulation node transmits serialized observation data to the edge node, while the edge node processes the input and returns the computed action values. This closed-loop interaction ensures synchronized execution between the simulated environment and the deployed model, allowing for the assessment of real-time inference performance on the edge device. By continuously exchanging serialized data packets over ethernet, PIL enables rigorous testing of the real-time feasibility, performance, latency,

and computational efficiency of deep reinforcement learning (DRL) models deployed onto edge devices. The detailed explanation of each node is discussed below:

4.2.1 Simulation Node. The simulation node hosts the CARLA environment, generating real-time semantically segmented (SS) images along with navigation-related data such as vehicle position, velocity, orientation, and lane adherence. These data collectively represent the vehicle's observation of its environment at each timestep. The simulation node performs data pre-processing and serialization of data to convert these observations into structured binary packets optimized for fast transmission over ethernet. In return, the simulation node receives the binary packets of action values computed by the edge node. These action values are unpacked and applied to the simulated vehicle within CARLA, effectively closing the loop for real-time control and enabling accurate evaluation of edge-based decision-making performance.

4.2.2 Prediction Node. This section outlines the deployment process of the VAE and the actor-network on edge device for real-time inference. The VAE and the actor-network receive serialized observation data from the simulation node. However, unlike the training framework, the image data is serialized to a linear array and packed into a byte format, which is suitable for transmission. Upon receiving this packed data, the edge node then unpacks it and reshapes it into the format suitable for the VAE, which is $160 \times 80 \times 3$. The VAE then encodes the image into a compact latent vector, which is the image observation, effectively capturing the essential visual features of the driving environment in a lower-dimensional representation. This latent vector is then concatenated with the navigation observation vector, forming a complete observation state. The actor-network is pre-trained and processes the observation state to generate mean action values. These values parameterize a Gaussian distribution defined using a fixed exploration noise, from which an action is sampled. The final action output is then serialized and transmitted back to the simulation node.

To enable the efficient deployment of training models on the edge devices, we incorporate post-training quantization techniques for both the Variational Autoencoder (VAE) and the actor-network. These compression methods are adopted to make the models computationally efficient for the edge. The uncompressed TensorFlow (TF) model is quantized to TensorFlow Lite (TFLite) models such as Floating-point16 (FP16) and Integer8 (INT8). These compression methods significantly reduce memory footprint and inference latency with negligible degradation of performance, enabling efficient and stable policy execution with adequate accuracy on low-power edge hardware.

5 EXPERIMENT SETUP AND RESULTS

The experiments are conducted on the server equipped with NVIDIA Quadro GPU. The software environment includes Python 3.7, CARLA 0.9.8, and TensorFlow 2.11. Moreover, for edge prediction, Raspberry Pi 4 Model B is used, which has the configuration of quad-core 64-bit ARM Cortex-A72 processor and 8GB RAM.

Hyperparameter settings: Table 1 summarizes the hyperparameters used for training the Variational Autoencoder (VAE). The Mean squared error (MSE) is chosen as the loss function as it effectively reduces the reconstruction error between the input and output images. The learning rate of 1×10^{-4} is selected to provide a balanced trade-off between convergence speed and training stability, preventing overshooting while allowing the model to progress during updates enhanced with Adam optimizer for its adaptive nature making it well-suited for deep neural networks. Moreover, a batch size of 64 with 10 epochs is applied to leverage mini-batch training, which improves generalization.

Similarly, in table 2, hyperparameters used for training the PPO-based RL agent are summarized. The training is conducted with a learning rate of 1×10^{-4} and an initial exploration noise of 0.4, chosen to encourage exploration in the early stages of training. This exploration noise is gradually

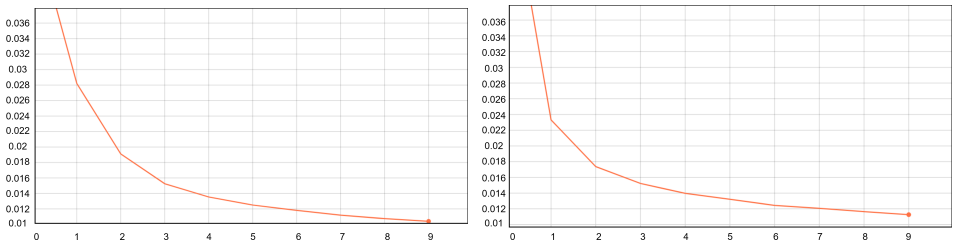
Table 1. Training Parameters for VAE

Hyperparameter	Value
Learning rate	1×10^{-4}
Batch size	64
Loss Function	MSE
Optimizer	Adam
Activation	ReLU
Latent space dimension z_{dim}	95
Epochs	10

Table 2. Training Parameters for RL Agent

Hyperparameter	Value
Initial Exploration Noise σ_{noise}	0.4
Advantage Estimation	GAE
Learning Rate	1×10^{-4}
Batch Size	1
Policy Clip ϵ	0.2
Discount Factor γ	0.99
Lambda λ	0.95

reduced over time to shift the agent's behavior toward exploitation as learning progresses. A batch size of 1 is used to accommodate the variable number of samples generated per episode in the interactive, episodic training process. Generalized Advantage Estimation (GAE) is employed with discount factor $\gamma = 0.99$ and smoothing parameter $\lambda = 0.95$, enabling the agent to compute more stable and informative advantage values by balancing bias and variance in advantage estimation, which is particularly beneficial in autonomous driving scenarios where long term rewards are impacted by present action.



(a) Training Loss (b) Validation Loss

Fig. 3. Training and validation loss of the VAE

Results:

Before experimenting with the RL agent, the VAE is trained using 12,000 SS images collected from 'town 02' in the CARLA simulator. The training is conducted using the hyperparameters listed in Table 1. During inference, only the encoder component of the VAE is utilized, as reconstruction through the decoder is not necessary for policy decision-making. The training performance of the VAE, as shown in figure 3, represents two graphs as training loss and validation loss. According to the graphs, both training and validation losses are steadily decreasing and remain within a desirable range, indicating that the model learns compact latent representations properly while minimizing the reconstruction error.

After the VAE, the RL agent is trained using the PPO algorithm, while keeping the weights of the encoder fixed to ensure consistent feature encoding. To train the RL model, we determine a 500-meter route in Town 2 of the CARLA simulator, as shown in Fig. 4, which includes elbow turns and straight road segments. At each intersection, the agent is directed to follow the straight path to maintain the route consistency. To execute this task, an interactive episodic approach is employed, guided by a dense reward function. In this function, rewards are defined by termination conditions to guide the learning process, particularly to prevent the policy from diverging by ensuring desired driving behaviors. In this setup, the episode termination criteria is defined below:

- Occurrence of infractions, such as collisions
- Deviation from the driving lane exceeds 3 m
- Exceeding the set velocity range 15 km/h to 35 km/h
- Reaching the maximum allowed episode length of 7500 timestep

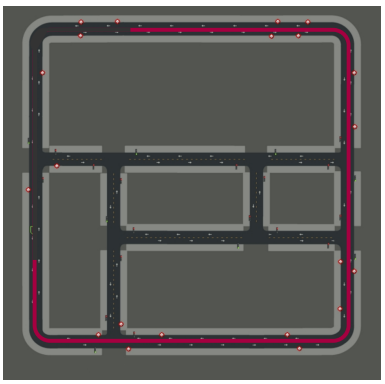
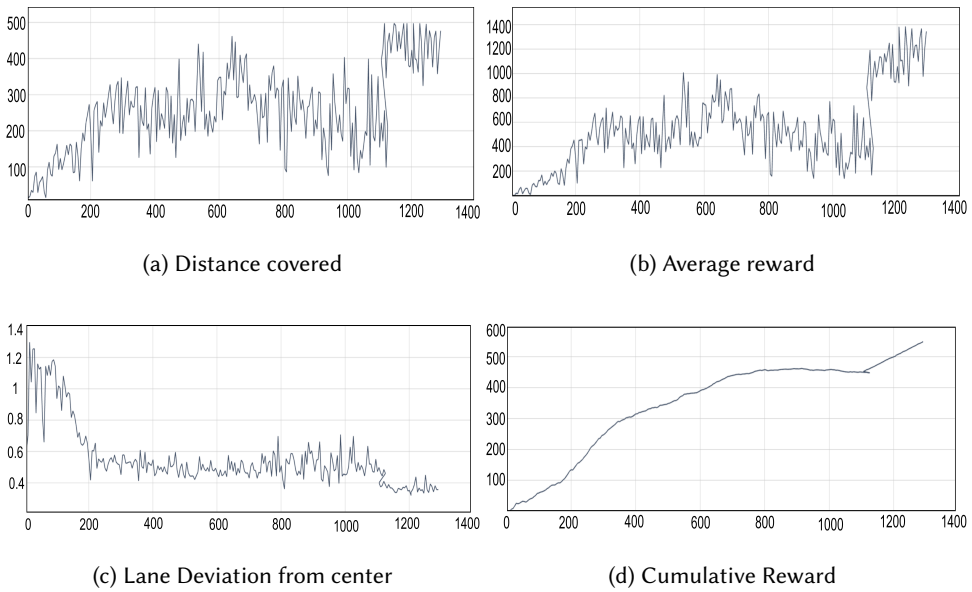


Fig. 4. Predetermined route of Town 2 in CARLA simulator



(a) Distance covered

(b) Average reward

(c) Lane Deviation from center

(d) Cumulative Reward

Fig. 5. Training Results of RL Agent on GPU per episode

Using these termination conditions, we train the agent for a total of 1200 episodes, with exploration noise gradually reduced by 0.05 for every 300 episodes from an initial value of 0.4. This scheduling enables a smooth transition from exploration to exploitation. To accelerate learning, a checkpoint is saved every 100 meters. If the agent reaches a terminal state—such as a collision or deviation from the lane—it is reset to the most recent checkpoint. This strategy enables the agent to rapidly return to challenging segments of the route, facilitating faster convergence and

focused learning in complex driving scenarios. As training progresses the average reward per episode and cumulative reward increase, while the lane deviation gradually decreases, as shown in Figure 5. Additionally, the PPO agent demonstrates the ability to reach the destination of 500 meters, confirming successful learning and policy convergence.

Table 3. Model Memory Requirements Under Different Quantization Levels

Model	***TF Model (float32)***	**Floating-point16 (*FP16*)**	**Integer8 (*INT8*)**
Actor	988 KB	456 KB	233 KB
Encoder	52.6 MB	26.8 MB	13.45 MB

After the training phase, we move towards the edge prediction by deploying the model on the edge. As mentioned before, we work on Raspberry pi 4 model B which has low memory and low processing head without GPU or TPU. As a result, it is difficult for real time inference while maintaining a high level accuracy. Therefore, we try to achieve a balance between accuracy and latency. To fulfill this goal we adopted the post-training quantization provided by tensorflow on both the DRL (only Actor network) and Variational Autoencoder (VAE) models. Given the limited memory resources of edge devices, minimizing model size is a key objective. Table 3 presents the memory requirements under different numerical precisions. Both models showed a reduction in size after quantization, with the Integer8 (*INT8*) version occupying the least memory. These results confirm the effectiveness of quantization in producing memory-efficient models suitable for real-time inference on edge hardware. To evaluate the performance of the quantized models, a dataset of 10 episodes is generated using the 32-bit GPU-trained model. Each episode contains semantic segmentation (SS) images, navigation data, the predicted output mean from the model, and the corresponding computational latency. This dataset is then used to test the *Floating-point16 (FP16)* and *Integer8 (INT8)* quantized models on the edge. For each episode, the inference time and the Root Mean Square Error (RMSE) between the outputs of the quantized models and the original 32-bit GPU model are recorded. This comparison helps us understand how well the quantized models perform in terms of both speed and accuracy.

Table 4. Root Mean Square Error for 10 episodes

	GPU	**Edge**	
Ep.	***TF-32***	***FP16***	***INT8***
1	0.02379	0.02379	0.02412
2	0.01469	0.01469	0.01500
3	0.00795	0.00795	0.00840
4	0.00252	0.00252	0.00363
5	0.00128	0.00128	0.00271
6	0.01585	0.01585	0.01613
7	0.00281	0.00281	0.00373
8	0.00000	0.00000	0.00302
9	0.00484	0.00484	0.00574
10	0.00272	0.00272	0.00352
Avg.	0.00764	0.00764	0.00860

Table 5. Inference Time for 10 episodes (in msec)

	GPU	**Edge**	
Ep.	***TF-32***	***FP16***	***INT8***
1	26.28	76.14	36.97
2	26.80	76.37	37.03
3	27.53	74.79	36.94
4	28.06	75.54	36.98
5	33.71	74.88	36.93
6	31.31	75.15	36.87
7	30.41	75.04	36.96
8	60.32	75.05	36.97
9	31.67	75.11	37.01
10	34.00	76.47	35.87
Avg.	33.01	75.45	36.85

The above tables demonstrate the comparison between the quantized models and the original TF-32 model, without any compression, in terms of inference time and accuracy. Table 4 and represents

the RMSE results of models before and after quantization. The Floating-point16 (*FP16*) model preserves the same accuracy as the original TF-32 model maybe because the reduced precision affecting the weights rather than model structure. However,*INT8* shows slight drop in accuracy which indicates the quantization error due to lower precision. Furthermore, the table 5 depicts the latency in the inference phase showing that the TF-32 model takes approx. 33 ms to process one sample in GPU while, *FP16* takes only 75 ms on a low-powered edge device without any GPU unit. In contrast, *INT8* takes approx. 36 ms to process one sample which shows comparatively very less inference time with adequate accuracy. Based on these results, the *INT8* quantized model is selected for the Processor-in-the-Loop (PIL) evaluation, as it provides an optimal trade-off between computational efficiency and prediction accuracy.

Table 6. GPU Prediction Results over 10 Episodes

Ep.	Time (s)	Reward	Dist. (m)	Latency (ms)	Speed (km/h)
1	100.48	1722.09	500	40.20	17.93
2	80.95	1364.26	395	40.36	17.57
3	102.07	1742.96	500	41.23	17.64
4	80.04	1289.90	393	42.28	17.68
5	101.50	1668.55	500	42.24	17.75
6	83.69	1276.41	400	43.87	17.21
7	100.48	1547.93	500	44.24	17.93
8	101.21	1561.83	500	44.98	17.78
9	100.17	1491.43	500	46.19	17.96
10	100.54	1380.66	500	49.18	17.89
Avg	95.91	1502.50	478.80	43.88	17.73

Table 7. Edge prediction Results with PIL over 10 episodes

Ep.	Time (s)	Reward	Dist. (m)	Latency (ms)	Speed (km/h)
1	95.52	1183.49	327	112.47	12.32
2	143.70	1455.82	500	113.88	12.50
3	95.14	952.43	319	107.91	12.07
4	150.56	1362.90	500	121.62	11.95
5	62.29	799.04	212	116.15	12.25
6	120.79	1276.91	472	129.97	14.04
7	83.57	940.33	287	123.95	12.36
8	127.90	1678.00	500	125.84	14.15
9	132.89	1568.00	500	108.74	13.61
10	118.08	1145.80	457	128.99	13.93
Avg	113.41	1236.97	407.4	118.45	12.92

In the edge prediction using PIL, the *INT8*-encoder model as well as the *INT8*-actor model are deployed onto the edge device and evaluated its performance by interfacing CARLA in a closed-loop. Action values are computed using the output of the actor model and standard deviation (σ_i) parameterized by exploration noise. This noise (σ_{noise}) is fixed at 0.2 during evaluation, corresponding to the

final value reached at the end of the training phase. Evaluation is carried out over 10 episodes with performance metrics such as distance covered, average speed, total time, reward per episode, and inference latency. These results are compared against the GPU baseline to assess the performance of *INT8* models. The overall loop latency for edge prediction averaged 118.45 milliseconds, with inference time being 36.85 milliseconds per sample, as shown in table 7. The agent achieved an average distance of 407.4 meters and an average reward of 1236.97 during these runs. In comparison with the GPU results illustrated in table 6, the agent deployed on the edge covered approximately 85.1% of the average distance and achieved around 82.3% of the average reward of GPU metrics. Notably, the overall loop latency on the edge is dominated by communication delay (81.6 ms), which is significantly higher than the model's computational inference time (36.85 ms). The above analysis shows that this work is executed on the edge, which includes the PIL setup. This signifies the models like DRL can be used for sequential decision-making in real time prediction.

6 DISCUSSIONS

The proposed framework connects the ADS simulation environment with a low-power edge device for real-time evaluation using a Processor-in-the-Loop (PIL) setup. The VAE-DRL model, trained with the PPO algorithm on a GPU, which is quantized further for deploying on the edge device to enable continuous sequential decision-making in a real-time feedback loop. The evaluation of this approach uses training performance metrics such as episode-wise reward, distance traveled, and completion time. However, this is to be noted that choosing the appropriate hyperparameters for training the RL agent turns out to be moderately complex. Particularly due to a variable number of samples per episode, which varies as training progresses, it also made it difficult to select a proper batch size. Moreover, the initial exploration noise is town-specific and depends on different road layouts and navigation complexities. During training in certain instances, the DRL model produced *NaN* values for the mean output, which is found to be caused by large values in the latent space generated by the VAE. To address this, value clipping is applied to the latent space representations.

In addition to the above issues, this paper raise a point regarding the compatibility of the quantized PyTorch model with ARM-based architecture. According to [16], initially we followed the GitHub repository that provided a PyTorch-based implementation of the DRL model, which is used for training. Nevertheless, during inference on the edge device, the *INT8* quantized PyTorch model is not able to run on the Raspberry Pi due to its incompatibility with ARM-based architecture. Due to these deployment challenges, we adopted to TensorFlow. It is important to note that, before performing real-time inference using the PIL, a dataset is generated using the GPU-trained model. This dataset is then used to run inference on the edge with quantized models.

Furthermore, moving onto the inference phase, prediction performance is assessed through RMSE loss and inference latency by comparing edge results with the GPU-based model, which served as the baseline model. Our variable exploration rate training strategy enabled a smooth transition from exploration to exploitation. As a result, the model is converged and exhibits driving behavior that is aligned with the defined objectives. The results from edge prediction using the PIL framework demonstrate that deploying DRL models on low-power edge devices for autonomous driving is feasible, as the computational latency of the *INT8* quantized model is comparable to that of the *TF32* model running on a GPU. Despite the communication delay introduced by the PIL loop, our DRL agent is still able to perform effectively, maintaining reliable decision-making and driving behavior during real-time inference on the edge device.

7 CONCLUSION AND FUTURE WORK

This paper explores and validates the feasibility of the practical deployment of DRL-based autonomous driving models on low-power edge devices using PIL evaluation. Merging Reinforcement

Learning with the autonomous driving domain by implementing it in the actual scenario and tackling important issues is the first work that provides the foundation for end-to-end prediction of deep reinforcement learning on the edge. Future work will focus on extending this framework to multi-agent systems and real-world vehicle integration to further assess the reliability of Deep Reinforcement Learning on the edge for autonomous driving. Moreover, since RL depends on server-based training, it introduces potential security concerns. Therefore, our aim is to provide a secure framework on the edge for RL implementation. We will also explore RL integration in a secure vehicle-to-vehicle (*V2V*) scenario in the *6G* network.

REFERENCES

[1] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016. Version 1, 25 Apr 2016.

[2] Felipe Codevilla, Matthias Müller, Alexey Dosovitskiy, Antonio López, and Vladlen Koltun. Learning by cheating. In *Conference on Robot Learning*, pages 1–15. PMLR, 2019.

[3] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. Carla: An open urban driving simulator. *Conference on Robot Learning (CoRL)*, pages 1–16, 2017.

[4] Sanskar Jadhav, Vedant Sonwalkar, Shweta Shewale, Pranav Shitole, and Samarth Bhujadi. Deep reinforcement learning for autonomous driving systems. *International Journal for Multidisciplinary Research (IJFMR)*, 6(5), September-October 2024. IJFMR240528518.

[5] Alex Kendall, James Hawke, David Janz, et al. Learning to drive in a day. *arXiv preprint arXiv:1807.00412*, 2018.

[6] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[7] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[8] Jens Kober, J. Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

[9] Jan Koutník, Jürgen Schmidhuber, and Faustino Gomez. Evolving deep unsupervised convolutional networks for vision-based reinforcement learning. *arXiv preprint arXiv:1406.6366*, 2014.

[10] Jesse Levinson, Jacob Askeland, Jonathan Becker, Jennifer Dolson, David Held, Sebastian Kammel, J. Zico Kolter, Dirk Langer, Oliver Pink, Vadim Pratt, and Sebastian Thrun. Towards fully autonomous driving: Systems and algorithms. *IEEE Transactions on Intelligent Transportation Systems*, 13(4):1355–1369, 2011.

[11] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[12] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Tim Harley, Timothy Lillicrap, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *International conference on machine learning*, pages 1928–1937, 2016.

[13] Ramesh Mohan, Robert Kolodziejski, and Henry Lee. Towards secure and efficient edge computing for autonomous driving: Challenges and solutions. *IEEE Vehicular Technology Magazine*, 14(2):27–35, 2019.

[14] Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. Importance estimation for neural network pruning. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11264–11272, 2019.

[15] Brian Paden, Michal Čáp, Sze Zheng Yong, Denis Yershov, and Emilio Frazzoli. A survey of motion planning and control techniques for self-driving urban vehicles. *IEEE Transactions on Intelligent Vehicles*, 1(1):33–55, 2016.

[16] Asad Idrees Razak. Implementing a deep reinforcement learning model for autonomous driving. *IEEE Transactions on Intelligent Transportation Systems*, 2024.

[17] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[18] John Schulman, Sergey Levine, Philipp Moritz, Michael Jordan, and Pieter Abbeel. Trust region policy optimization. *arXiv preprint arXiv:1502.05477v5*, Apr 2017. Department of Electrical Engineering and Computer Sciences, University of California, Berkeley.

[19] John Schulman, Philipp Moritz, Sergey Levine, Michael I Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. In *International Conference on Learning Representations (ICLR)*, 2016.

[20] Wilko Schwarting, Javier Alonso-Mora, and Daniela Rus. Planning and decision-making for autonomous vehicles. *Annual Review of Control, Robotics, and Autonomous Systems*, 1:187–210, 2018.

[21] Rishabh Sharma and Prateek Garg. Optimizing autonomous driving with advanced reinforcement learning: Evaluating dqn and ppo. *IEEE*, 2024. IEEE Xplore Part Number: CFP24V90-ART.

[22] Weisong Shi, Jie Cao, Qun Zhang, Youhu Li, and Lanyu Xu. Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, 3(5):637–646, 2016.

[23] Sanjna Siboo, Anushka Bhattacharyya, Rashmi Naveen Raj, and S. H. Ashwin. An empirical study of ddpg and ppo-based reinforcement learning algorithms for autonomous driving. *arXiv preprint*, November 2023. Corresponding author: Rashmi Naveen Raj (rashmi.naveen@manipal.edu), Supported by Manipal Academy of Higher Education.

[24] Alexander Stens, Iversen Szewczyk, Frank Lindseth, and Gabriel Kiss. Ai-agents trained using deep reinforcement learning in the carla simulator. *Journal of Autonomous Systems Research*, 2022.

[25] Fei Wang, Meng Jiang, Kai Qian, et al. Efficient and interpretable neural attention mechanisms for autonomous driving. *IEEE Transactions on Neural Networks and Learning Systems*, 31(11):4859–4872, 2019.

[26] Bernhard Wymann, Eric Espié, Christian Guionneau, et al. Torcs, the open racing car simulator. *Open-Source Software*, 4(1), 2014.

[27] Yiming Zeng, Zhongli Yan, and Shuang Wang. Distributed deep learning framework for autonomous vehicles with edge computing. *IEEE Access*, 7:102108–102116, 2019.