# Deep Reinforcement Learning on Edge: A Feasibility Study for Autonomous Driving

Thummalabavi Sankshay Reddy

*Under the guidance*
*of*
Dr. Ayantika Chatterjee

Indian Institute of Technology Kharagpur

May 5, 2025

# Outline

# Outline (continued)

# Autonomous Driving Systems

- An Automotive Driving System (ADS) integrates advanced technologies such as sensors, AI, functional algorithms, and communication networks to enable autonomous vehicle perception, decision-making, and control.
- Traditional ADS implementations use rule-based systems, such as semi-automatic adaptive cruise control and automatic braking systems. Current research is increasingly focusing on fully automated driving systems using machine learning techniques.

**Need for Autonomous Driving systems?**

- **Safety:** Reducing human errors is the reason for most traffic accidents.
- **Efficiency:** Enhancing traffic flow and reducing congestion.
- **Environmental Impact:** Optimizing routes to lower emissions and fuel consumption.
- **Economic Benefits:** Reducing transportation costs and boosting industries like logistics and ride-sharing.
- **Accessibility:** Providing mobility solutions for individuals unable to drive.

# Limitations of Rule-Based Models

Rule-based systems operate using predefined logic, relying on expert knowledge to encode traffic laws, obstacle avoidance, and navigation strategies. They are often implemented using state machines, decision trees, or logic-based controllers.
**Limitations :**

- **Modular but Isolated:** While the modular design simplifies individual tasks, proper module integration limits overall system adaptability in complex, dynamic environments.
- **Rigid and Inflexible:** Cannot adapt to unpredictable scenarios in dynamic environments.
- **Scalability Challenges:** Adding new rules increases system complexity.
- **Limited Generalization:** Performs poorly in situations not explicitly predefined.

**To overcome these challenges, modern Autonomous Driving systems are using Machine Learning techniques like Deep Learning and Reinforcement Learning.**

# Role of Machine Learning in Autonomous Driving

- **Scalability:** Rule-based systems require exhaustive programming for every scenario, making them impractical for the complexity of real-world driving.
- **Adaptability:** ML-based systems learn from data and improve over time, allowing them to handle novel or unexpected driving conditions more effectively.
- **Perception Complexity:** ML algorithms, especially Deep Learning, can process raw sensor data like camera images and LIDAR to detect objects and interpret the environment.
- **End-to-End Learning:** ML enables integrated learning from perception to control, reducing manual pipeline tuning and improving system cohesion.

# Some of the key ML techniques for ADS

| Model/Technique | Description | Limitations |
|---|---|---|
| **Convolutional Neural Networks (CNNs)** | Implemented by NVIDIA [1], to map raw pixels from camera inputs directly to steering commands. Demonstrated end-to-end learning for autonomous driving. | Extensive dataset requirements, limitations of deep learning, and high-end GPUs needed; significant barriers for edge computing environments. |
| **Deep Q-Networks (DQN)** | Demonstrated remarkable efficiency in tasks like playing Atari games and robotic control [5]. | Struggles with continuous action spaces or high-dimensional input data. Can become unstable or diverge due to complexities. |
| **Deep Deterministic Policy Gradient (DDPG)** | Implemented using an Actor-Critic model with 4 neural networks and a replay buffer. Proven to be stable and optimal [6]. | Complex architecture with multiple networks and a replay buffer is not ideal for edge computing. High memory usage and computational overhead. |
| **Proximal Policy Optimization (PPO)** | Implemented using Actor-critic model without any reply buffer. Proven to be stable and simple compared to DDPG [2]. | Exploration vs Exploitation in training, heavyweight for Edge Deployment, Complex Hyperparameter Tuning. |

Table: Some of the key ML techniques for ADS

- Deep Learning techniques like DNN and CNN are more effective for classification and feature extraction tasks, but heavily depend 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.
- Complementing traditional Deep Learning models like DNNs or CNNs with Reinforcement Learning techniques enhances interactivity, enabling better decision-making in dynamic autonomous driving scenarios.
- Currently, many autonomous driving systems are increasingly adopting DRL due to their interactive nature and inherent reward-punishment mechanisms.

Numerous methods and techniques are employed in autonomous driving, ranging from traditional rule-based systems to advanced machine learning algorithms. Despite the progress made, each method has limitations that must be addressed.

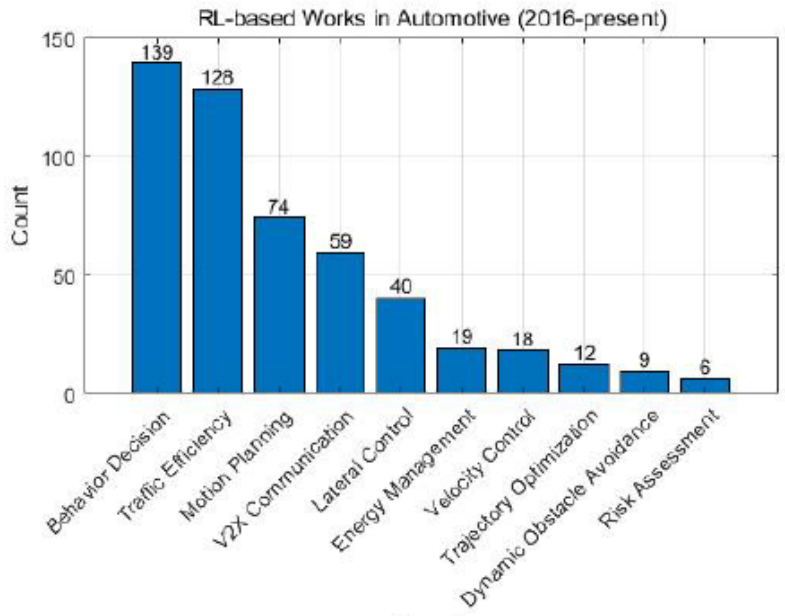



Figure: Survey: DRL in Path Planning and Control (2024)

# Key Terminology

1. **Agent** The decision-making entity that interacts with the environment by observing states, taking actions, and receiving rewards.
2. **Policy** A strategy used by the agent to determine its actions based on the current state, typically denoted as $\pi(a|s)$, representing the probability of taking action $a$ in state $s$.
3. **Model** The model typically refers to a neural network approximating functions such as the policy in deep reinforcement learning. It is trained using collected experience to guide the agent's behavior.
4. **Episode** A complete sequence of interactions between the agent and the environment, starting from an initial state and continuing until a terminal state is reached or a time limit is exceeded.
5. **Environment** The simulated world in which the agent operates and learns. This work is an autonomous driving simulator that requires substantial computational resources due to its complex physics and high-resolution rendering.

# Targeted Challenges in V2I

- This work focuses on advancing Vehicle-to-Infrastructure (V2I) communication for autonomous driving by addressing key challenges that impact safety and efficiency.
- **Targeted challenges include:**
  - **Collision avoidance:** Preventing accidents through predictive and reactive decision-making.
  - **Lane deviation minimization:** Ensuring the vehicle stays centered in the correct lane, essential for structured road environments.
  - **Maintaining a safe velocity range:** Regulating speed to adapt to traffic and environmental conditions.
  - **Optimal path planning:** Navigating the vehicle efficiently to reach its destination with minimal delays and safe maneuvers.
- **Goal:** To develop a learning-based control framework that enables the vehicle to operate autonomously while satisfying all the above safety and navigation requirements.
- A Deep Reinforcement Learning (DRL) method, **Proximal Policy Optimization (PPO)**, is adopted for learning robust and safe driving policies.

# Overview

The proposed system is divided into :

- **Simulation Setup** – CARLA-based virtual driving environment.
- **Training the Agent** – PPO-based DRL training using VAE and Actor-Critic architecture on GPU.
- **Post Training Quantization** - Enables faster and more energy-efficient execution on edge devices, making it ideal for real-time autonomous driving applications.
- **Prediction using Processor-in-the-Loop (PIL)** – Real-time inference on edge device using a lightweight Actor network. Enable closed-loop testing between the simulation and the edge node. Validate the feasibility of deploying DRL models in real-time on low-power edge hardware.

# DRL-Based PIL Framework

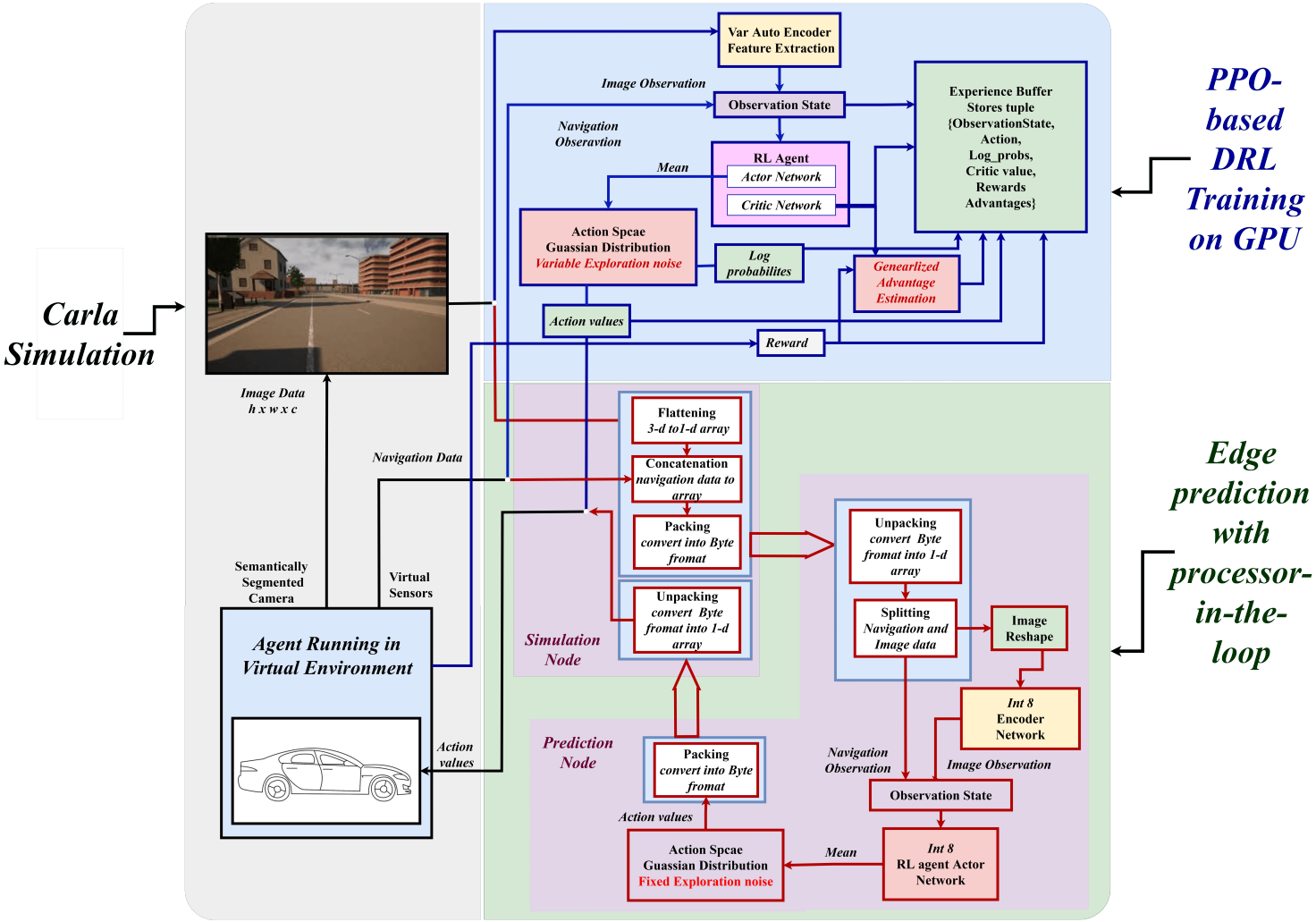



Figure: PPO-based DRL Prediction within the Processor-in-the-Loop Framework

# CARLA Simulator

CARLA (Car Learning to Act) is an open-source autonomous driving simulator developed to support the development, training, and validation of autonomous driving systems. It provides high-fidelity urban environments, realistic traffic scenarios, and a flexible API that enables researchers to simulate sensor data, control vehicles, and test algorithms under diverse weather and lighting conditions. CARLA is widely used for safe and scalable experimentation in perception, planning, and control tasks.

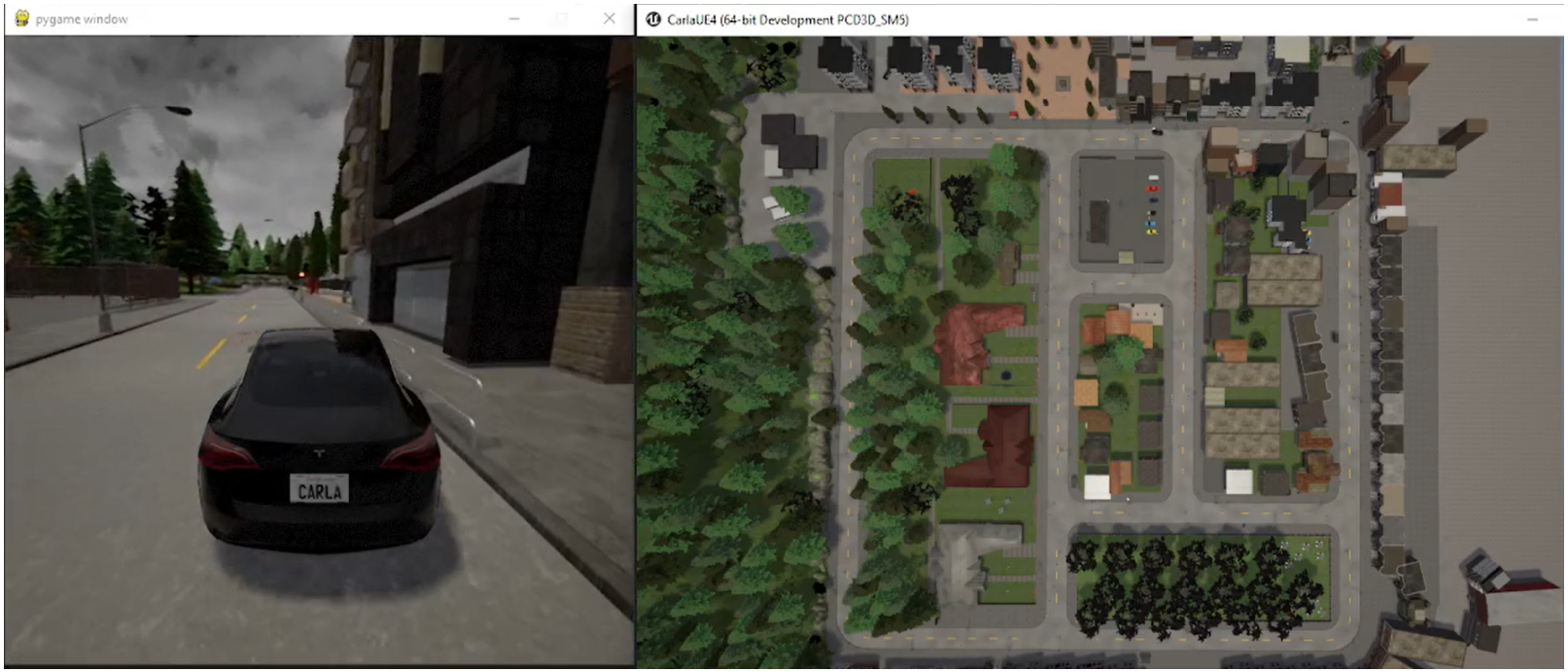



Figure: Vehicle and Top view of Carla simulator

# PPO-Based DRL Training on GPU

## Training Model VAE and DRL

- The training pipeline integrates VAE for feature extraction and Proximal Policy Optimization (PPO) for decision-making in the CARLA simulation environment.
- **VAE** for semantic feature extraction (lane markers, pedestrians, Obstacles, etc.)
- **DRL Agent**: Actor-Critic architecture for sequential decision-making, trained using modified PPO algorithm

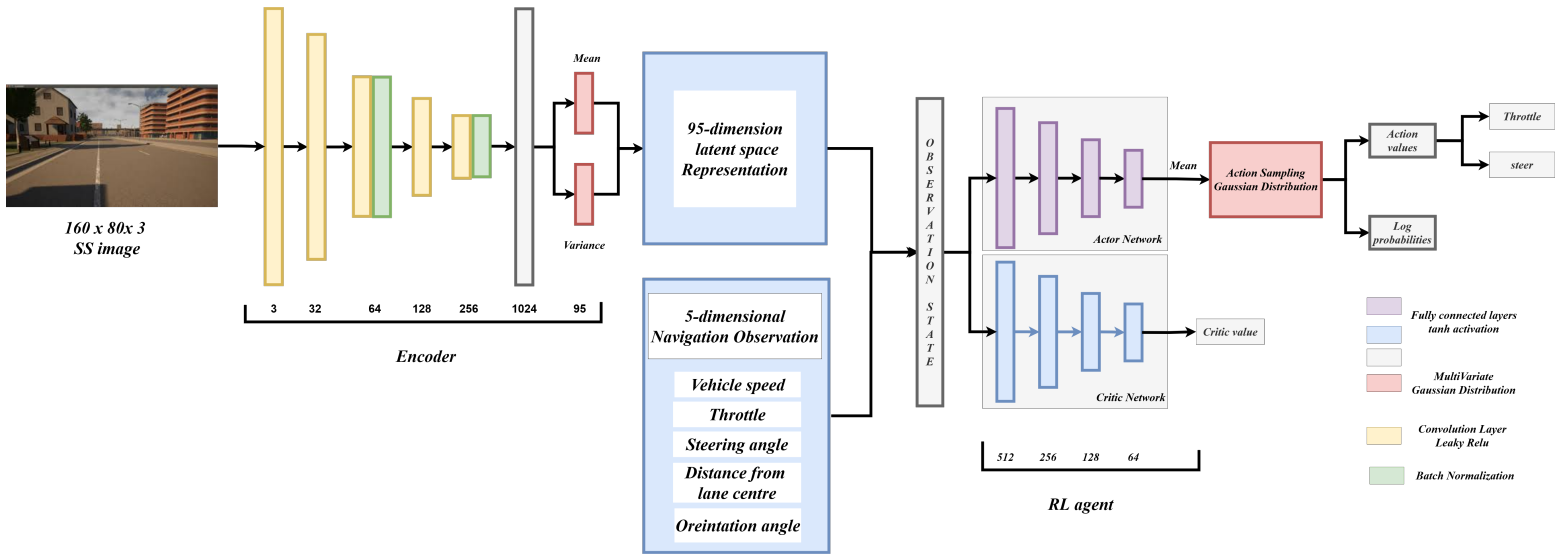



Figure: Training Model (VAE integrated with RL agent)

# Why Proximal Policy Optimization ?

Proximal Policy Optimization (PPO) offers several advantages that make it particularly suitable for sequential decision-making tasks in Autonomous Driving Systems (ADS):

- **Stability in Learning:** PPO uses a clipped surrogate objective that restricts large policy updates, providing stable and reliable learning, essential for safety-critical applications like autonomous driving.
- **Sample Efficiency:** By enabling multiple epochs of mini batch updates from the same trajectory data, PPO improves sample efficiency, which is beneficial in simulation-heavy ADS training environments such as CARLA.
- **On-Policy Learning:** PPO uses data collected by the current policy, allowing tighter feedback loops and policy alignment, which is important when adapting to rapidly changing traffic dynamics.

# Proposed PPO for Autonomous Driving

- Initialize policy $\pi_\theta$, value function $\pi_\phi$, exploration noise $\sigma_{\text{noise}}$, and buffer $B$
- For each iteration $k = 1, \ldots, K$:
  - If $k$ mod $p = 0$, decrement $\sigma_{\text{noise}}$ by a factor of $x$
  - For each episode $n = 1, \ldots, N$:
    - Collect trajectories using old policy $\pi_{\theta_{old}}$
    - Store transitions $(s_t, a_t, r_t, V(s_t), d_t)$ in buffer $B$
  - Compute advantage estimates $\hat{A}_t$ using GAE and value targets $V_{\text{target}}$
  - For each update step $i = 1, \ldots, n_{\text{updates}}$:
    - Compute sampling ratio $r_t(\theta)$
    - Compute policy loss $L_{\text{policy}}$
    - Compute gradients and update parameters
  - 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 $B$

# Variable Exploration Noise ($\sigma_{\text{noise}}$)

- Introduced a time-varying exploration noise $\sigma_{\text{noise}}$ to shift from exploration to exploitation during training.
- Actions are sampled from a Gaussian distribution:

$$a_t \sim \mathcal{N}(\mu_\theta(s_t), \Sigma_\theta), \quad \Sigma_\theta = \text{diag}(\sigma_i)$$

- Initially, higher $\sigma_{\text{noise}}$ increases randomness in actions; gradually reduced to refine policy behavior. Used to promote diverse action sampling early and stabilize decision-making as learning progresses.

# Generalized Advantage Estimation (GAE)

The thesis adopts the GAE[4] advantage function, which guides the direction and magnitude of policy updates by quantifying the relative value of actions to improve the accuracy of advantage calculations during policy training.
In the GAE equation:

$$\hat{A}_t = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}, \quad \delta_t = r_t + \gamma V_\phi(s_{t+1}) - V_\phi(s_t) \tag{1}$$

- $V_\phi(s_t)$ is the value function, which predicts the total future reward starting from state $s_t$.
- $\gamma$ (Discount Factor): controls how much future rewards matter; higher values look further ahead, lower values focus on immediate rewards.
- $\lambda$ (GAE Parameter): helps balance between learning fast (low $\lambda$) and learning accurately with more stability (high $\lambda$).

This balance is important because it prevents the model from changing too quickly or too slowly, leading to smoother and better learning over time.

# Loss Function

**Loss Function** [2] is defined as the aggregation of clipped surrogate loss, value loss, and an entropy regularization term.

$$L(\theta) = L^{\text{CLIP}}(\theta) - c_1 L^{V}(\theta) + c_2 L^{S}(\theta) \tag{2}$$

where coefficients $c_1$ and $c_2$ are weighting the value loss and entropy, respectively.

**Clipped Surrogate Objective** [3]

$$L^{\text{CLIP}}(\theta) = \mathbb{E}_t \left[ \min \left( r_t(\theta)\hat{A}_t,\ \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right] \tag{3}$$

with:

- $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$: the probability ratio between the new and old policies.
- $\epsilon$: a hyperparameter (typically 0.1 or 0.2) that constrains the update step.

This formulation ensures conservative policy updates by penalizing excessive deviations from the previous policy.

**Value Function Loss** [2] To ensure accurate estimation of state values, PPO includes a squared-error loss term for the value function:

$$L^V(\theta) = \mathbb{E}_t \left[ \left( V_\theta(s_t) - V_t^{\text{target}} \right)^2 \right] \tag{4}$$

where $V_\theta(s_t)$ is the predicted value and $V_t^{\text{target}}$ is typically the empirical return $\hat{R}_t$.

**Entropy Bonus** [2] To promote sufficient exploration during training, an entropy regularization term is added:

$$L^S(\theta) = \mathbb{E}_t \left[ \beta \cdot \mathcal{H}(\pi_\theta(\cdot|s_t)) \right] \tag{5}$$

where $\mathcal{H}$ denotes the entropy of the probability distribution and $\beta$ is a scaling coefficient.

# Trajectory Collection and Policy Update

- PPO requires fresh data at each iteration for stable updates, unlike Q-learning, which reuses experience. Hence, trajectories are collected **on-policy at runtime**, using the current (or slightly old) policy.
- A trajectory consists of:
  - State $s_t$
  - Action $a_t$
  - Reward $r_t$
  - Value $V(s_t)$
  - done $d_t$
- Trajectories are collected on an episodic basis. The episode ends when termination conditions (e.g., collisions, lane deviation, or max episode length) are met.
- Agent runs in CARLA (Town 2) environment using old policy $\pi_{\theta_{\text{old}}}$.
- Data for each trajectory is stored in a buffer after each episode.
- Advantage estimates are computed; policy and value losses are calculated using the trajectory data from the buffer.
- Gradients are used to update actor and critic networks, and the policy is refreshed.

# Edge Prediction with Processor-in-the-Loop (PIL)

- **Simulation–Edge Integration:** CARLA simulator runs on high-performance platforms (GPU/cloud), while observations are sent to a low-power edge device (e.g., Raspberry Pi) via Ethernet RPCs; the edge performs inference and returns actions in real-time, forming a synchronized closed-loop control system.
- **Why PIL:** Allows early testing of latency, real-time feasibility, and performance of DRL models on constrained hardware before actual deployment.

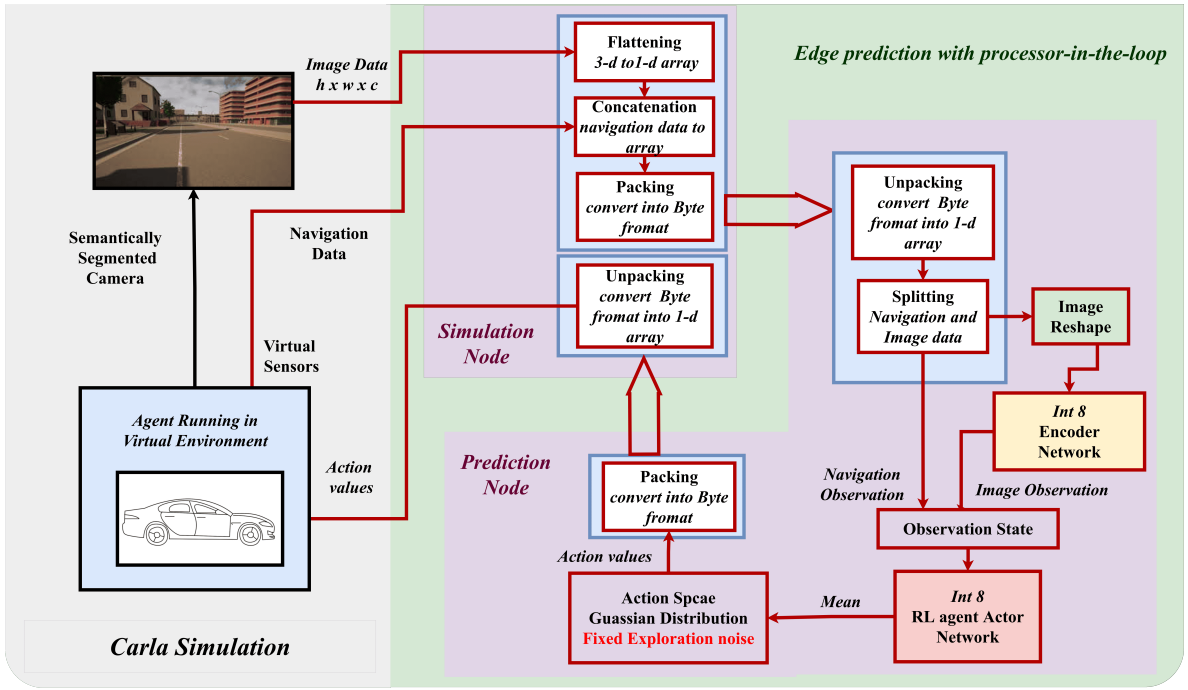



Figure: Prediction using Edge device (PIL)

# Simulation Node (CARLA)

- Hosts the CARLA simulator on high-performance systems (e.g., GPU servers or cloud platforms).
- Generates real-time semantic segmentation (SS) images and navigation data (position, velocity, lane info).
- Packages image and navigation data into structured serialized byte arrays for efficient transmission.
- Sends serialized observations to the edge node via Ethernet using remote procedure calls (RPC).
- Receives serialized action values from the edge, deserializes them, and applies them to control the vehicle.
- Enables a closed-loop real-time control system for assessing inference performance on edge devices.

# Prediction Node (Edge Device)

- Runs on low-compute platforms (e.g., Raspberry Pi or Jetson Nano) for real-time inference.
- Receives and unpacks serialized observations from the simulation node.
- Reshapes image data into $160 \times 80 \times 3$ format suitable for VAE input.
- Uses VAE to encode images into a compact latent vector representing visual features.
- Concatenates the latent vector with navigation data to form a complete observation state.
- Feeds the observation state into a pre-trained actor-network to generate action values, which are serialized and sent back.

# Hyper parameters

Table: Training Parameters for VAE

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

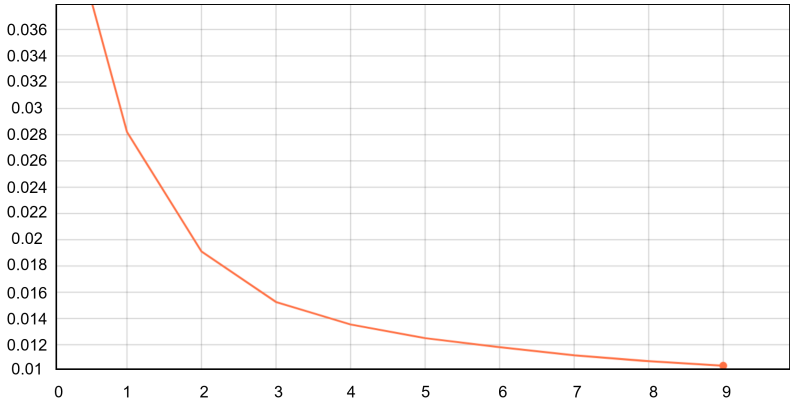
- Mean Squared Error (MSE) is used as the loss function to minimize the difference between input and reconstructed images.
- Learning rate of $1 \times 10^{-4}$ ensures stable convergence without overshooting.
- Adam optimizer is used for its adaptive learning capability, suited for deep networks.
- A batch size of 64 is chosen to leverage mini-batch training for improved generalization and trained over 10 epochs to balance computational efficiency and learning progress.

Table: Training Parameters for RL

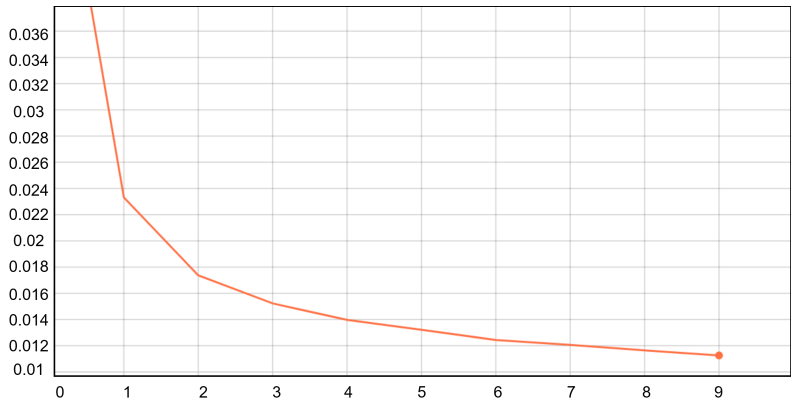
| **Hyperparameter** | **Value** |
|---|---|
| Initial Exploration Noise $\sigma_{\text{noise}}$ | 0.4 |
| Advantage Estimation | GAE |
| Learning Rate | $1 \times 10^{-4}$ |
| Batch Size | 1 |
| Policy Clip $\epsilon$ | 0.2 |
| Discount Factor $\gamma$ | 0.99 |
| Lambda $\lambda$ | 0.95 |

- Learning rate of $1 \times 10^{-4}$ is used to ensure stable convergence during training.
- Initial exploration noise of 0.4 is set to encourage exploration in early training stages. Exploration noise is decayed gradually to promote exploitation as training progresses.
- Batch size of 1 is chosen due to the variable number of samples generated per episode.
- Policy clip value of 0.2 is used to stabilize updates, as recommended in PPO [3].
- GAE [4] is applied with discount factor $\gamma = 0.99$ for long-term reward capture. GAE smoothing parameter $\lambda = 0.95$ balances learning speed and stability.

# Results




(a) Training Loss




(b) Validation Loss

Figure: Training and validation loss of the VAE

- The VAE is trained using 12,000 semantically segmented (SS) images from 'town 02' in the CARLA simulator.
- Only the encoder is used during inference for generating latent representations, while the decoder is excluded during inference since reconstruction is not needed for policy decisions.
- Figure 6 shows training and validation loss curves. Both losses decrease steadily and stay within a desirable range, indicating proper learning of compact features.

# RL Agent Training Setup

- PPO algorithm is used to train the RL agent, keeping the VAE encoder weights fixed for consistent feature extraction.
- An interactive, episodic training approach uses a dense reward function.
- Rewards are tied to termination conditions to guide learning toward desired driving behavior.
- Episode termination criteria:
  - Collision
  - Lane deviation greater than 3 m
  - Velocity outside the range 15 km/h to 35 km/h
  - Episode exceeds 7500 timesteps

- A dense reward function $R$[2] trains the agent toward safe and efficient driving behavior. The reward $R$ considers three factors: the agent's speed, lane alignment, and infractions.

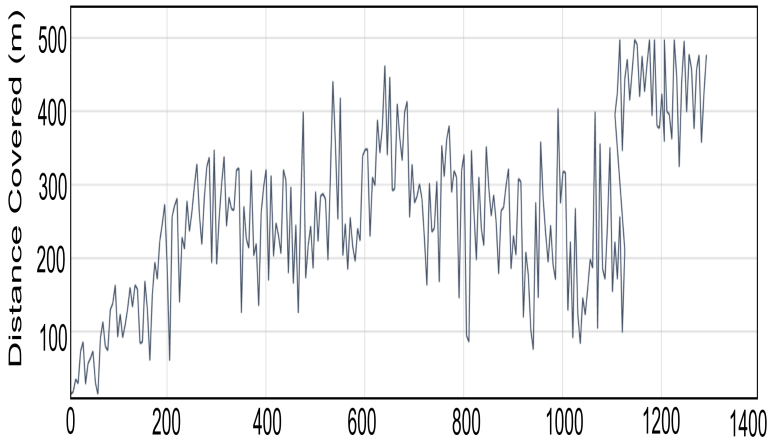
$$R = \begin{cases} -N_o, & \text{on infraction} \\ v_{min} \times (1 - d_{norm}) \times R_o, & v < v_{min} \\ 1 \times (1 - d_{norm}) \times R_o, & v_{min} \leq v < v_t \\ (1 - \frac{v - v_t}{v_{max} - v_t}) \times (1 - d_{norm}) \times R_o, & v \geq v_t \end{cases} \tag{6}$$

- Lane alignment is evaluated using the distance between the vehicle's center and the lane center. This distance is normalized as $d_{norm} = \frac{d}{d_{max}}$, where $d_{max}$ is a threshold from termination conditions.
- This orientation term ensures the agent maintains the correct heading for effective steering, and the orientation reward $R_o$ is computed as:

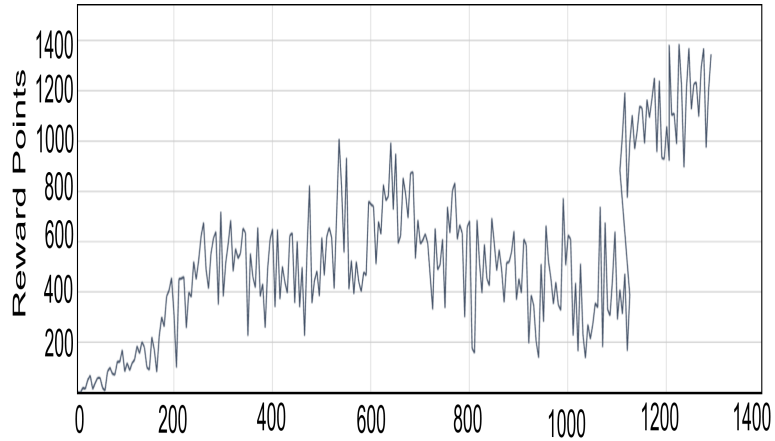
$$R_o = \begin{cases} 1 - \frac{|\alpha_{diff}|}{\alpha_{max}}, & \text{if } \alpha_{diff} < \alpha_{max} \\ 0, & \text{otherwise} \end{cases}$$

- The agent is incentivized to stay near the lane center, drive at optimal speed, and maintain orientation to maximize reward while avoiding infractions.

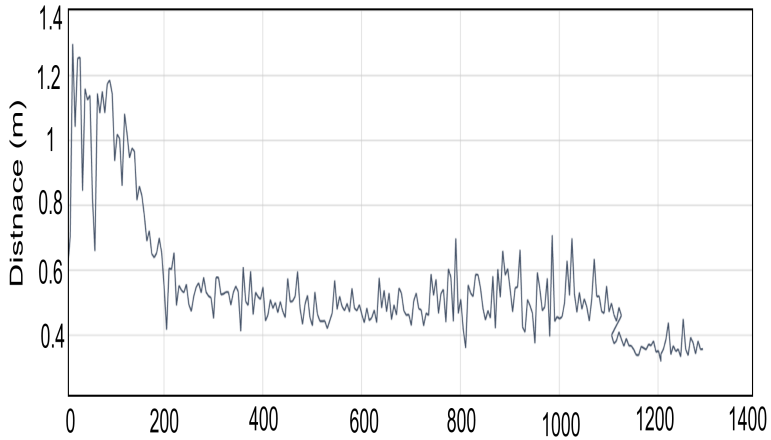
- The agent is trained over 1200 episodes using defined termination conditions.
- Exploration noise starts at 0.4 and is reduced by 0.05 every 300 episodes to shift from exploration to exploitation.
- A checkpoint is saved every 100 meters to allow quick resets and faster learning. This checkpointing strategy accelerates learning in complex segments by focusing on difficult areas.
- Upon reaching a terminal state (e.g., collision or lane deviation), the agent resets to the latest checkpoint.
- Over time, average and cumulative rewards per episode increase, while lane deviation decreases, as shown in Figure 7.
- The PPO agent completes the 500-meter route, indicating policy convergence and effective learning.



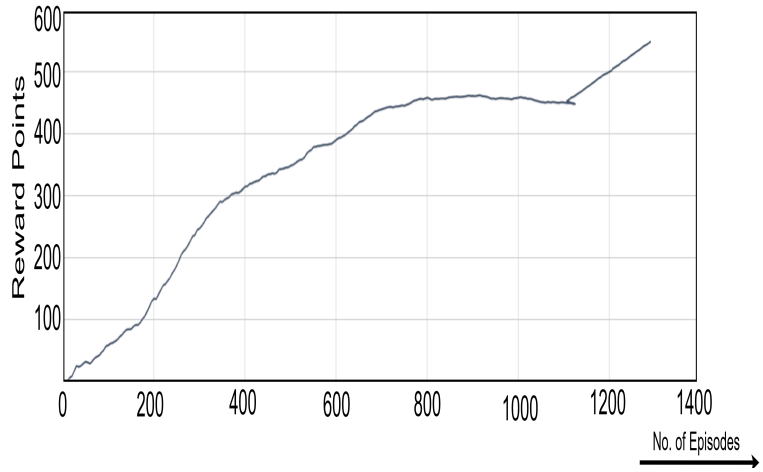

(a) Distance covered




(b) Average reward




(c) Lane Deviation from center




(d) Cumulative Reward

Figure: Training Results of RL Agent on GPU per episode

- The deployment is performed on a Raspberry Pi 4 Model B, which lacks GPU/TPU support and has limited memory and processing power.
- To address this, TensorFlow's post-training quantization is applied to both the Actor network of the DRL model and the VAE.
- Reducing model size is essential due to the limited memory capacity of the Raspberry Pi. Table 4 shows that quantization significantly reduces model size, with the INT8 format yielding the smallest memory footprint.

Table: 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 |

- To assess the quantized models, a dataset of 10 episodes is generated using the 32-bit GPU-trained model.
- This evaluation helps compare the speed (latency) and accuracy (RMSE) of the quantized models. The analysis provides insight into the trade-offs between performance and efficiency on resource-constrained edge hardware.

Table: Root Mean Square Error

| | 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: Inference Time (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 |

Given its low latency and acceptable accuracy, the *INT8* model is chosen for PIL deployment as the best trade-off between speed and performance.

- In the Processor-in-the-Loop (PIL) evaluation, both the *INT8*-encoder and *INT8*-actor models are deployed on the edge device.
- These models interface with CARLA in a closed-loop system for real-time action prediction and control.
- The actor model computes action values, and the standard deviation $\sigma_{\text{i}}$ is derived from a fixed exploration noise $\sigma_{\text{noise}} = 0.2$, consistent with the final training phase.
- Evaluation is conducted over 10 episodes to assess model performance in realistic scenarios.
- Key performance metrics recorded include distance covered, average speed, total time to reach the target distance, reward per episode, and end-to-end inference latency.
- Results from the *INT8* deployment are compared with the GPU-based baseline to evaluate the effectiveness and efficiency of the quantized models on edge hardware.

Table: GPU Prediction Results over 10 Episodes

| Ep. | Total Time (s) | Reward | Dist. (m) | Latency per action (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 |

Evaluation is conducted over 10 episodes to assess model performance. Key performance metrics recorded include distance covered, average speed, total time to reach the target distance, reward per episode, and end-to-end inference latency.

Table: Edge prediction *int8* Results with PIL over 10 episodes

| Ep. | Total Time (s) | Reward | Dist. (m) | Latency per action(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 Processor-in-the-Loop (PIL) evaluation, both the *INT8*-encoder and *INT8*-actor models are deployed on the edge device. These models interface with CARLA in a closed-loop system for real-time action prediction and control. The actor model computes action values, and the standard deviation $\sigma_{\text{i}}$ is derived from a fixed exploration noise $\sigma_{\text{noise}} = 0.2$, consistent with the final training phase.

- The average overall loop latency for edge prediction was **118.45 ms**.
- The average inference time per sample was **36.85 ms**, as shown in Table 8.
- The agent achieved an **average distance of 407.4 meters** and an **average reward of 1236.97** during edge deployment.
- Compared to GPU results (Table 7):
  - The edge agent achieved **85.1%** of the GPU's average distance.
  - It attained **82.3%** of the GPU's average reward.
- The **communication delay** (81.6 ms) accounted for the majority of the loop latency, significantly more than the inference time.
- These results confirm that **edge deployment using PIL** enables real-time sequential decision-making with DRL models.

| Paper | Exp. Setup | Algo. | Targets | Edge | Time | Comments |
|---|---|---|---|---|---|---|
| [1] | NVIDIA DevBox | CNN | Lane Following | - | - | Used CNN for autonomous steering control; Neural network used is complex |
| [5] | CARLA simulator | DQN | Collision Avoidance and path efficiency | - | - | Comparison between DQN and PPO in various Scenarios, Performance metrics- Reward |
| [6] | CARLA Simulator | DDPG | Lane Following, Collision Avoidance, braking | - | - | Empirical analysis on continuous control, Performance metrics include episodic Reward |
| [2] | CARLA and NVIDIA GPU | PPO | Lane Following, Collision Avoidance, set Velocity Range, Path efficiency | - | - | PPO based DRL training on High end GPU, Performance metrics include episodic Reward, lane deviation, distance traveled |
| **This Work** | CARLA, NVIDIA Quadro, Rpi, PIL | PPO | Lane Following, Collision Avoidance, set Velocity Range, Path efficiency | Rpi | yes | Prediction with deployed model on EDGE in PIL loop, Performance metrics include episodic Reward, lane deviation, distance traveled, RSME, Inference time, Overall latency(comp + Comm. time) across various TensorFlow and lite models |

# Conclusion

- The proposed framework integrates an ADS simulation environment with a low-power edge device using a **Processor-in-the-Loop (PIL)** setup for real-time evaluation.
- A **quantized VAE-DRL model**, trained with PPO on a GPU, is deployed on the edge device to perform sequential decision-making in a closed-loop system.
- Evaluation is based on key training performance metrics such as **episode-wise reward**, **distance traveled**, and **completion time**.
- **Hyperparameter tuning** is moderately complex due to the varying number of samples per episode as training advances.
- Selecting an appropriate **batch size** is challenging because of the dynamic episode lengths.
- The initial **exploration noise** is town-specific and varies with different road layouts and navigation complexities.
- In some training runs, the DRL model produced **NaN outputs** for the action mean, traced to **large latent values** from the VAE.
- This issue is mitigated by applying **value clipping** to the latent space representations.

- An additional challenge addressed in this thesis is the **incompatibility of the quantized PyTorch model** with ARM-based architecture.
- As noted in [2], the initial training followed a GitHub repository implementing the DRL model in **PyTorch**.
- However, the *INT8* quantized PyTorch model **failed to run on the Raspberry Pi** due to architectural incompatibility.
- These deployment limitations prompted the switch to **TensorFlow**, which offered better support for ARM-based edge devices.
- Prior to real-time inference using the **Processor-in-the-Loop (PIL)**, a dataset was generated using the **GPU-trained model**.
- This dataset was used to test inference on the edge device with the quantized TensorFlow models.

- In the inference phase, prediction performance is evaluated using **RMSE loss** and **inference latency**, with the GPU-based model serving as the baseline.
- The variable exploration rate during training facilitated a **smooth transition from exploration to exploitation**, resulting in policy convergence.
- The trained model demonstrates **goal-aligned driving behavior**, confirming successful learning.
- **Edge prediction using the PIL framework** shows that DRL models can run feasibly on low-power edge devices.
- The *INT8* quantized model achieves **latency close to the TF32 GPU baseline**, despite the added communication delay from the PIL loop.
- The DRL agent maintains **reliable, real-time decision-making and driving behavior** during edge inference.
- This work confirms the **practical deployment feasibility** of DRL-based autonomous driving models on edge devices using PIL evaluation.
- It is one of the **first efforts to bridge deep reinforcement learning with edge deployment** for autonomous driving in a closed-loop, real-time setting.

# Future Work

- Extend to multi-agent and real-world vehicle testing.
- Explore on-device training for real-time adaptability.
- Use trainable exploration noise for better convergence.
- Combine PPO with DDPG for improved action stability.
- Expand action space (e.g., braking) for richer control.
- Employ attention/transformer models for temporal learning.

# Publication Update

## Journal Submitted to CASES 2025

We have submitted our research in the domain of autonomous driving and edge intelligence to the **CASES 2025** track at **Embedded Systems Week (ESWEEK)**, under the **Machine Learning** domain.

**Title**

*"Feasibility of Deep Reinforcement Learning on EDGE For Autonomous Driving"*

**Authors:**

1st Thummalabavi Sankshay Reddy

2nd Sumansmitha Rout

3rd Dr. Ayantika Chatterjee

*Advanced Technology Development Centre, IIT Kharagpur*

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

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

[4] 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.

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

[6] 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.

# Thank You!

Questions and Discussions Welcome