RL Algorithms in Autonomous Driving

CSE 546: Reinforcement Learning

Aniket Kumar

Ashutosh Sharan

Xueyi Yang

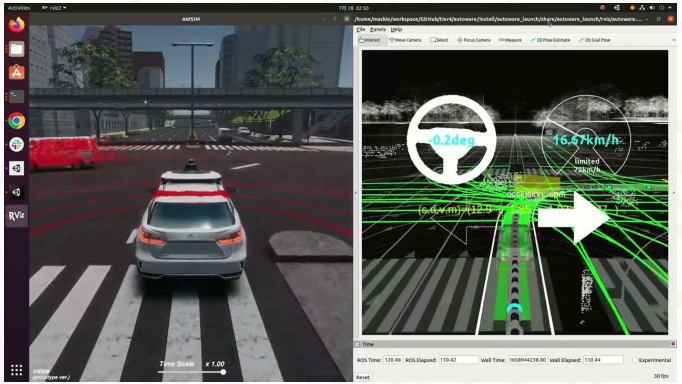
University at Buffalo The State University of New York

Problem Description

- **How can reinforcement learning algorithms be effectively applied to train autonomous agents to drive safely and realistically in complex traffic environments using the MetaDrive simulator?**
- **Challenges**
  - Navigate roads and intersections safely
  - Avoid collisions with vehicles/objects
  - Reach destinations efficiently
  - Continuous action space (steering, acceleration, braking)

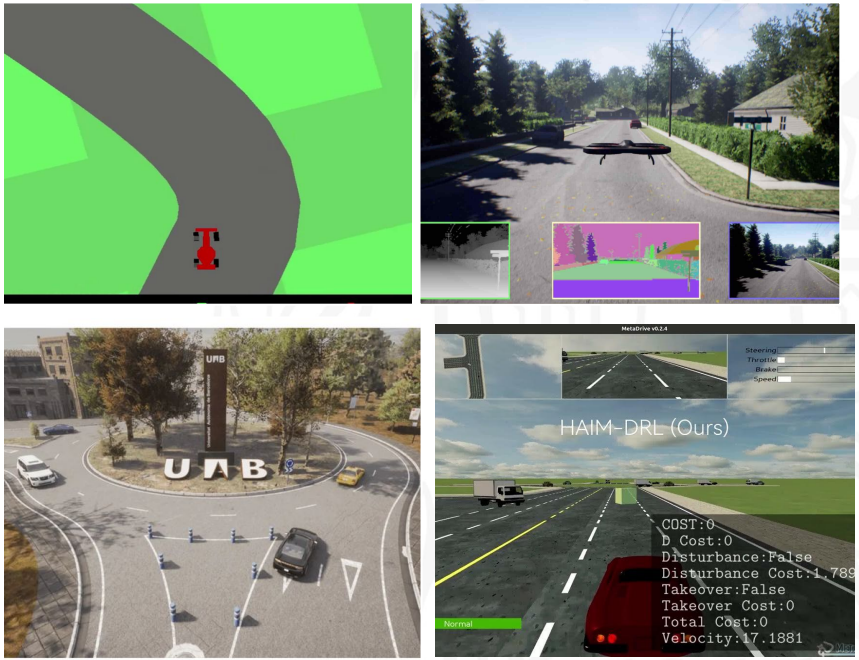
Background: RL in Autonomous Driving

- Autonomous driving requires decision-making in complex, dynamic environments.
- Traditional rule-based systems struggle with scalability and adaptability.
- Reinforcement Learning (RL) offers a data-driven approach to learn optimal driving policies through trial and error.
- Key challenges include safe exploration, generalization across scenarios, and balancing multiple driving objectives.



Related Work & Simulation Platform

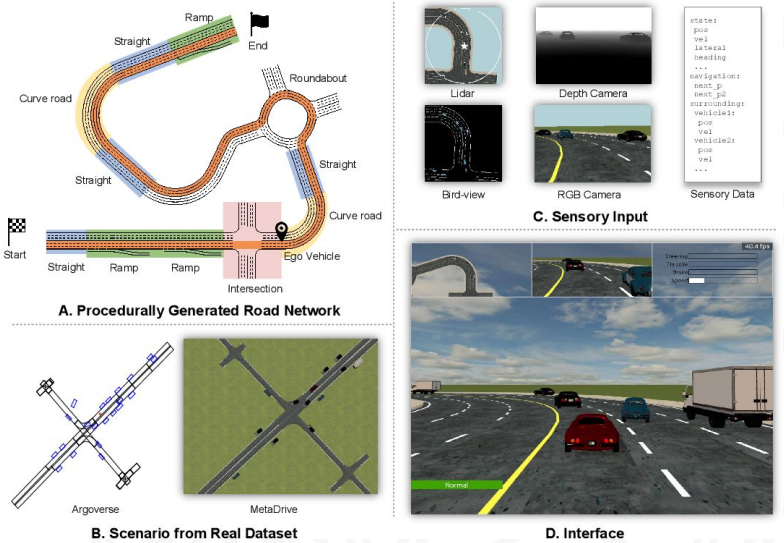
- Early works used OpenAI Gym (e.g., CarRacing-v0) but lacked realism and map diversity.
- Carla and AirSim offer high-fidelity simulations but are computationally expensive.
- MetaDrive is a lightweight simulator built for RL research:
  - Supports procedurally generated maps.
  - Offers many sensor input: LiDAR, camera etc.
  - Enables scalable training on diverse driving tasks.



About the Environment

MetaDrive is a lightweight, modular, and procedurally generated driving simulator tailored for reinforcement learning (RL) research in autonomous driving. It enables the creation of diverse driving scenarios by composing various road segments which facilitates the generation of an infinite number of unique driving environments, promoting the development of generalizable driving policies.

The simulator supports realistic physics and offers multiple sensor modalities, including LiDAR, RGB cameras, and top-down semantic maps, providing rich observational data for agents. Its lightweight design ensures high simulation speeds, making it accessible for training RL models on standard computing hardware.



About the Environment

Key Features:

Procedural Map Generation:

Utilizes symbolic strings (e.g., "OSX") to compose maps:

- O: Roundabout
- S: Straight road
- X: Intersection

Facilitates the creation of intricate driving environments for robust agent training.

Set num_scenarios=100 to generate 100 unique driving scenarios, promoting agent generalization across diverse environments.

Sensor Modalities:

LiDAR: Provides a 240-dimensional array representing distance measurements around the vehicle.

RGB Camera: Offers first-person visual input, simulating real-world driving perspectives.

Top-Down View: Supplies a bird's-eye perspective, useful for evaluation and debugging.

TopDownMetaDrive wrapper : Employed the wrapper to utilize top-down observations during training.

Methods

Algorithm	Type	Source	Action Space	Observation Type
DQN	Value-based	SB3	Discrete	Vector Input
PPO	Policy gradient	SB3	Continuous	Vector Input
Dueling DQN	Value-based (custom)	Custom	Discrete(18)	LiDAR(240-D)
A2C	Actor-Critic	CNN	Continuous	Top-down image
PPO	Policy gradient	CNN/Fusion	Continuous	Top-down image, Vector Input, LiDAR(120-D)

Dueling DQN

In this experiment, we implemented an Dueling Double DQN (DDQN) agent for autonomous driving in the MetaDrive environment using LiDAR-based observations and a discretized continuous action space. We introduced custom reward structure to encourage safe, smooth, and goal-directed behavior, including penalties for lane deviation, collisions, and sharp steering, especially near intersections. The training pipeline featured prioritized experience replay, a linear learning rate scheduler, and frame-stacked inputs to improve temporal awareness. Performance was evaluated periodically, with rewards and policy stability improving over episodes. Training metrics such as episode rewards, epsilon decay, and evaluation scores were logged and visualized to monitor learning progress.

Dueling DDQN Architecture:

Feature Layer: • Input: 750D (240 LiDAR + 10 features × 3 frames)

- Layers: 750 → 512 → 512 → 256
- LayerNorm + ReLU for stability and non-linearity
- Purpose: Extracts high-level features (e.g., obstacle proximity, lane alignment)

Value Stream: • Layers: 256 → 256 → 128 → 1

- BatchNorm + ReLU Output:V(s)(statevalue)

Advantage Stream: • Layers: 256 → 256 → 128 → 18

- BatchNorm + ReLU • Output: A(s, a) (action advantages)
- Purpose: Differentiates action benefits (e.g., steer left vs. right)

Q-Value Computation:

- Q(s, a) = V(s) + (A(s, a) − mean(A(s, a)))
- •Combines streams subtracts

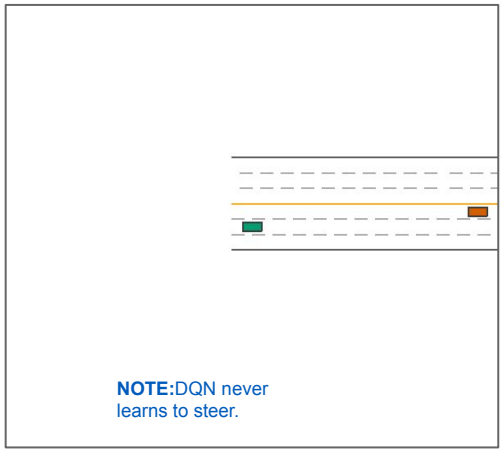
Dueling DQN Results

Due to the continuous nature of MetaDrive's action space, we discretized it to apply Dueling DQN, which only supports discrete actions. However, this discretization introduces limitations in fine-grained control. In particular, the agent tends to accelerate through intersections without appropriate braking or steering adjustments. This is because the fixed set of discrete actions lacks combinations needed for smooth, context-aware driving — such as gradually slowing down or executing soft turns. As a result, the agent often fails to respond appropriately in complex scenarios like intersections, leading to suboptimal or unsafe behaviors.

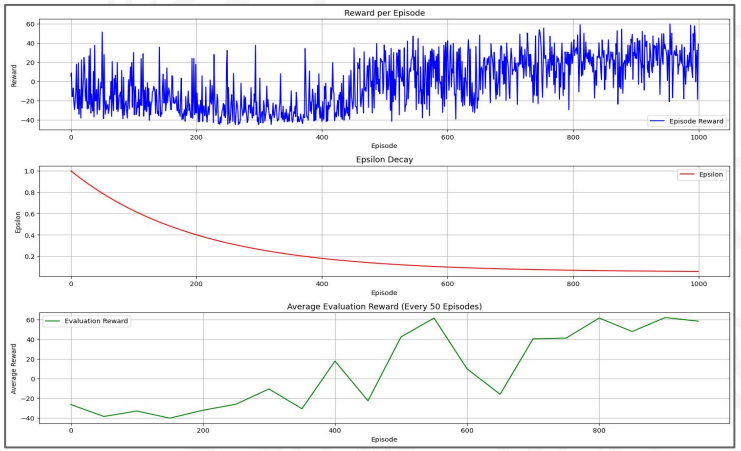
1. Custom Reward Function:

Driving Reward: 2.5· progress · lateral factor

- Progress: Longitudinal distance traveled
- Lateral factor: 1 − (lateral)^3, penalizes lane deviation
- Encourages forward movement while staying centered
- Speed Reward: 0.5· speed (halved in intersections)
- Speed: Normalized to 70% max speed
- Reduced in intersections for safety



NOTE:DQN never learns to steer.



Stable-Baselines3 DQN

Tried using Stable-Baselines3 DQN as a performance baseline.

Trained on the OSX map (roundabout + straight + intersection).

Issue: Default DQN struggled with steering, especially in curves.

Solution: Introduced a custom reward function to guide learning.

Conclusion:

DQN underperforms in this setting due to:

Discrete action space – too coarse for smooth driving.

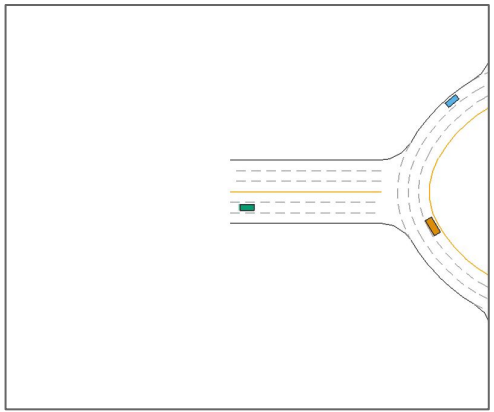
Off-policy Q-learning – unstable in dynamic, continuous-like domains.

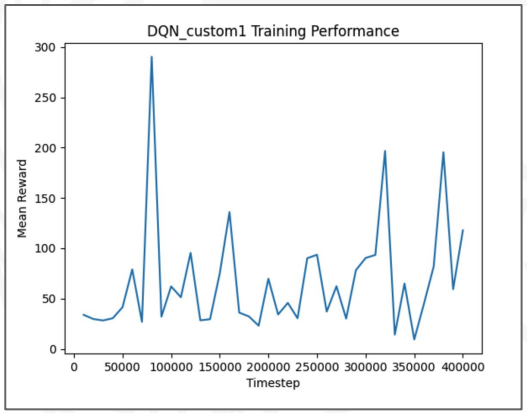
:

Component	Value	Purpose
Center driving reward	+0.3	Stay near lane center (offset < 0.5m)
Overspeed penalty	−1.0	Penalize speed > 20 km/h
Curve control bonus/penalty	±0.2 / −0.5	Encourage slowing for curves
Intersection heading reward	±0.3 / −0.5	Reward proper angle at intersections
Destination proximity bonus	+0.4	Reward for nearing the goal
Steering smoothness	±0.1 / −0.2	Penalize jerky turns
LiDAR-based overtaking reward	±0.2 / −0.5	Reward safe passing (if LiDAR enabled)
Sidewalk crash penalty	−4.0	Heavily punish off-road crashes

Stable-Baselines3 DQN Results:

A Stable-Baselines3 DQN agent trained on the MetaDrive OSX map using custom safety-focused rewards showed reliable lane-following and basic intersection handling.In certain training runs, the agent successfully entered curves — but only when the reward penalized sharply for poor behavior and we manually capped the agent's speed under 20 km/h.





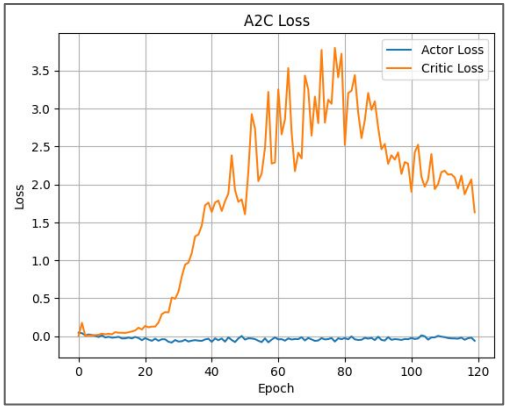
A2C(image obs based CNN network)

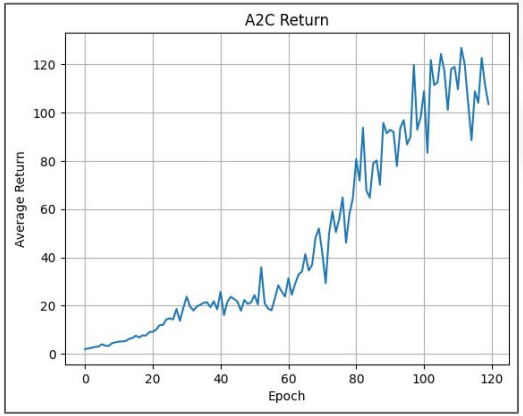
We trained our A2C agent using 5-channel top-down semantic images as input, where each channel represents **road layout, planned route, past ego positions,** and **nearby traffic information**. These images are passed through **convolutional layers** in both the **actor** and **critic** networks. The **actor** network outputs two continuous values: **steering and throttle**, representing the agent's actions. The **critic** network outputs a **single scalar value** estimating how good the current state is. To train the model, we use a **critic loss** based on **mean squared error** between predicted and actual returns, and an **actor loss** that maximizes the probability of good actions using **advantage-weighted log-probabilities**. Both networks are updated using gradient descent, with the advantage values computed using Generalized Advantage Estimation (GAE) to reduce variance and improve learning stability.

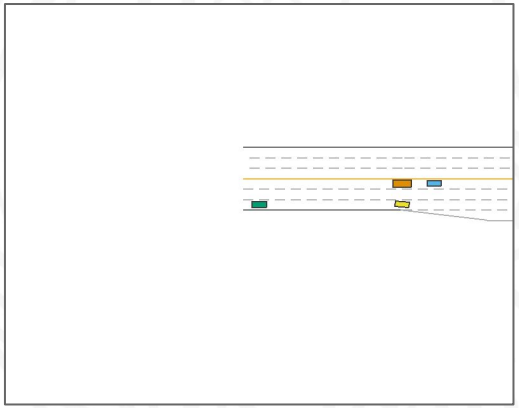
Issue	A2C Behavior	Real-World Effect
High update variance	No clipping → unstable learning	Jerky, inconsistent driving decisions
Sample inefficiency	Discards data after one update	Slower learning, poor generalization
No policy constraints	Can overcorrect on each update	Forgetting learned behaviors
Critic instability	Rising loss → poor value estimates	Bad action selection; misjudged risks/rewards

A2C Results

A2C trains steadily, but without mechanisms to control update size or reuse data efficiently, it is less suited for high-dimensional, precision-control tasks like autonomous driving with vision inputs. PPO or PPO3 (with clipping and multimodal input) handles these challenges better.







Advantages of PPO over A2C:

- It uses a clipped update rule to stabilize learning.
- It allows multiple gradient steps per batch to improve efficiency.

PPO (image obs based CNN network)

Actor Network:

Convolutional layers:
Conv2d(5→16, kernel=8, stride=4) → ReLU
Conv2d(16→32, kernel=4, stride=2) → ReLU
Flatten → Fully Connected:
Flatten → Linear(32×9×9=2592 → 256) → ReLU
Output:
Mean (μ) for 2D action (steering, throttle)
log_std as a trainable parameter
Final action = tanh(μ + std * ε), ensuring output ∈ [-1, 1]

Critic Network:

Shares the exact same convolutional backbone as the actor.
Fully connected layer outputs a single scalar value estimation.

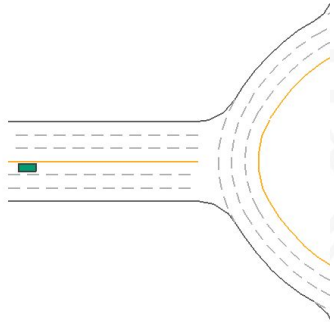
PPO (CNN) Implementation details and results analysis:

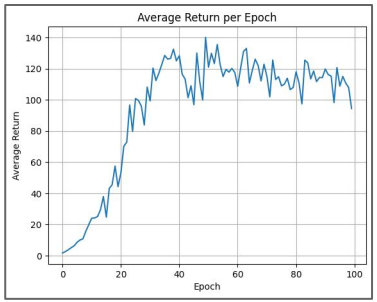
We used a 5-channel top-down semantic image with shape (5, 84, 84). (Road layout, Navigation route, Past ego positions, Current traffic, Previous traffic)

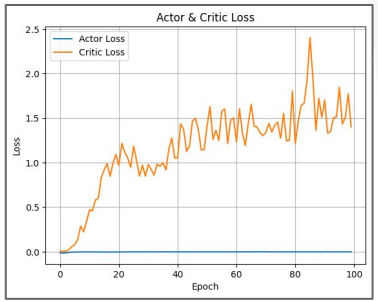
We squash the action output through tanh.

We used an epoch-based training mode, with full episode rollouts followed by PPO updates.

This setup gives us better control over the training process and more stable optimization.







PPO (image + vector obs based fusion network)

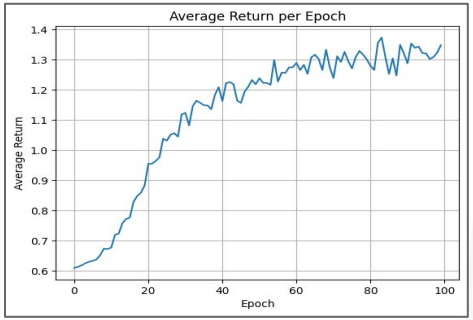
Component	Details
Image Encoder	`Conv2d(5→16, 8×8, stride=4)` → ReLU → `Conv2d(16→32, 4×4, stride=2)`
Image FC	`Linear(32×9×9 = 2592 → 256)` → ReLU
Vector FC	`Linear(5 → 64)` → ReLU
LiDAR FC	`Linear(120 → 128)` → ReLU
Fusion Layer	`Linear(256 + 64 + 128 = 448 → 256)` → ReLU
Output Head	Actor: `μ ∈ ℝ²`, `log_std` (trainable) Critic: Scalar value estimate
Action Space	2D continuous actions (steering, throttle), bounded with `tanh` ∈ [−1, 1]

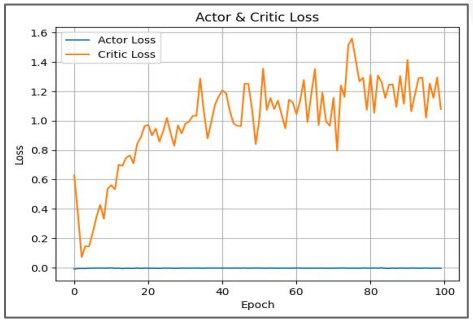
PPO (Fusion) Implementation details and results analysis:

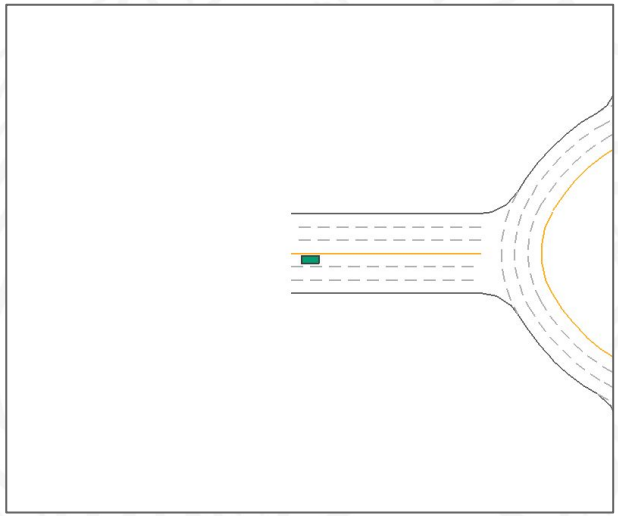
We upgraded the model to PPO3, which adds vector + LiDAR(120-beam) input data on top of the image.

The vector input includes:

- Ego speed
- Lane offset
- Distance to front vehicle
- Relative speed to front car
- Heading alignment

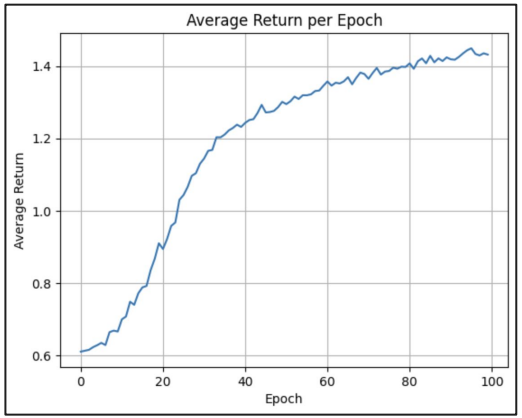






PPO

Result in Straight-Line Env without traffic



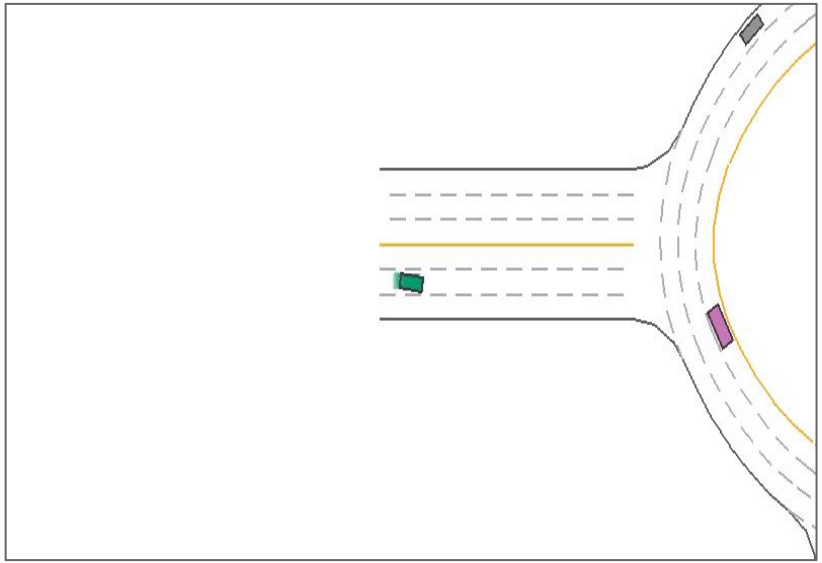
Methods

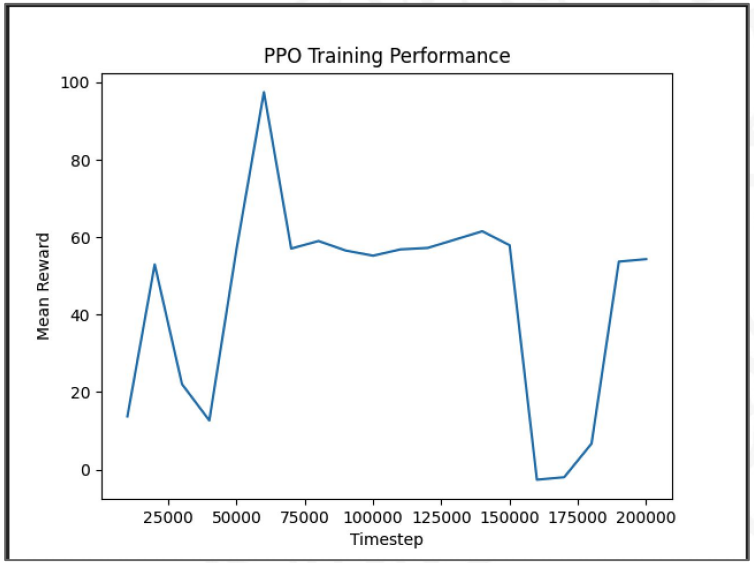
Stable Baseline PPO:

We approached the autonomous driving task by training a PPO agent using Stable-Baselines3 on the MetaDrive OSX map. The agent interacts with a continuous control environment and is guided by a reward function embedded in the simulator that incentivizes progress and penalizes unsafe behavior. We used an MLP-based policy network and trained the agent for 200,000 timesteps with regular evaluations every 10,000 steps to monitor performance. To visualize learning progress, top-down driving videos were recorded every 50,000 steps. The entire process was made reproducible using a fixed random seed, and training performance was tracked through reward plots and saved model checkpoints.

Results

Stable Baseline PPO:





Summary

In summary, the study highlighted that while value-based RL methods like DQN can be adapted with tweaks, continuous control methods (like PPO, A2C) are better suited for the complexities of autonomous driving in MetaDrive.

Key Strengths of Our Implementation

- **Multi-modal Perception**:Combined image, vector, and LiDAR inputs allow richer observation of the driving environment, capturing both semantic context and spatial distances.
- **Modular and Reproducible Design**:Clear separation of training, evaluation, and video logging pipelines enables consistent debugging, reproducibility, and flexible architecture upgrades.

Limitations in Our Current Approach

- **Weak Collision Avoidance**:Even with enhanced inputs and reward shaping, the agent fails to reliably learn obstacle-avoidance or car-following behavior in dense traffic.
- **Reward Dependence and Policy Collapse**:The agent's final performance remains highly sensitive to the reward design. Unstable reward feedback can lead to late-episode crashes and policy instability

Contribution

Aniket Kumar - Dueling DDQN from scratch with Lidar and RGB camera, and DQN from Stable Baseline implementation

Ashutosh Sharan - A2C from scratch on TopDownMetaDrive environment with image input and PPO from Stable Baseline implementation

Xueyi Yang - PPO from scratch with different input, custom reward structure for SB3 DQN and PPO, and Stable Baseline PPO

References

- ***https://metadrive-simulator.readthedocs.io/en/latest/***
- ***Prof. Alina CSE546 Class Lecture***

Thank You !!