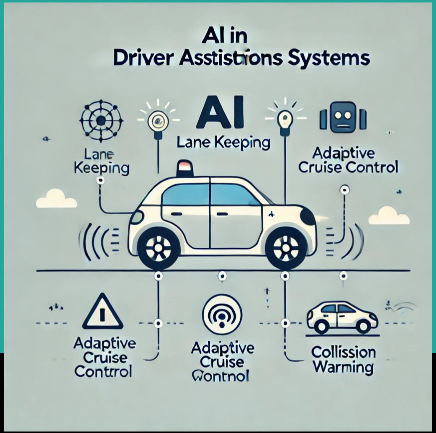


AI-Powered Adaptive Cruise Control

AN OVERVIEW OF THE INTEGRATION OF AI IN ADAPTIVE CRUISE CONTROL SYSTEMS FOR ENHANCED AUTOMOTIVE SAFETY AND EFFICIENCY.

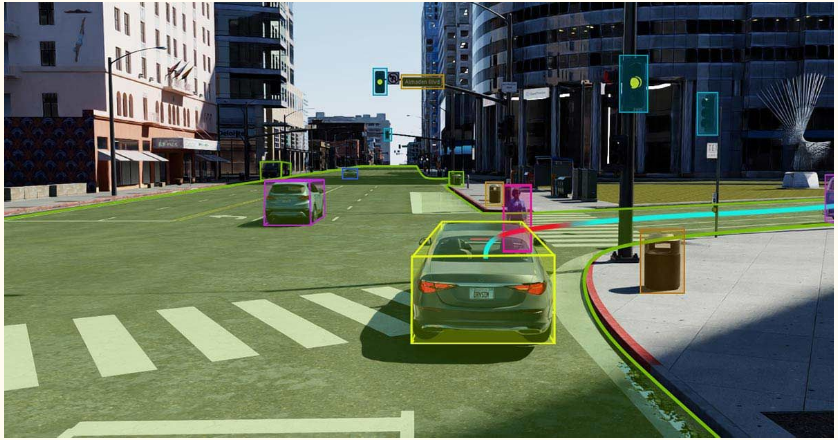
Introduction

This project demonstrates a fully autonomous vehicle simulation with the AI based Adaptive Cruise Control Implementation using CARLA simulator.

The vehicle autonomously manages its braking and throttling while maintaining a safe distance from the lead vehicle. It learns these behaviors through a RL model.

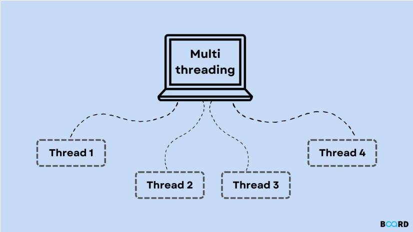
Key features of the simulation include:

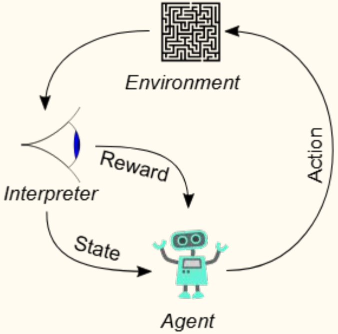
- Real-time sensor integration
- Adaptive driving mechanisms
- Traffic management



Technologies Used

- **Python:** Core programming language for integration and simulation logic.
- **CARLA Simulator:** A robust open-source platform for autonomous driving research.
- **Reinforcement Learning (RL):** Utilized for training the adaptive cruise control mechanism.
- **Computer Vision and Sensor Fusion:** Processes RGB camera feeds and LiDAR point clouds for real-time perception.
- **Multithreading:** Manages the spectator camera and vehicle-following logic concurrently.





Overview of Vehicle Setup

- **Vehicle Initialization and Management:** Spawns an ego vehicle in CARLA's virtual environment, provides functionality to spawn and manage a lead vehicle for ACC purposes.

- **Sensor Integration:** Attaches a range of sensors to the vehicle for perceiving the environment. These include:
  - Semantic LiDAR
  - RGB Front Camera
  - IMU (Inertial Measurement Unit)
  - GNSS (Global Navigation Satellite System)

- **Data Processing:** Processes sensor data in real-time, such as converting camera images to NumPy arrays or extracting key LiDAR points for obstacle detection. Employs callbacks to handle sensor outputs dynamically.

- **Environment and Traffic Interaction:** Enables synchronous simulation mode for deterministic behavior. Actively manages the traffic environment using CARLA's Traffic Manager to set desired speeds, lane changes, and safety buffers.

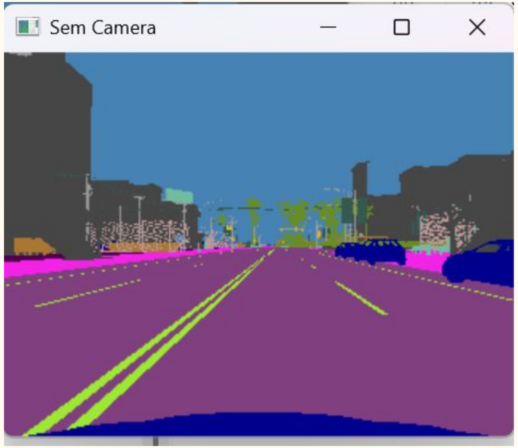
- **Spectator Camera:** Implements a top-down spectator camera that dynamically follows the vehicle during the simulation.

Sensors and Data Acquisition

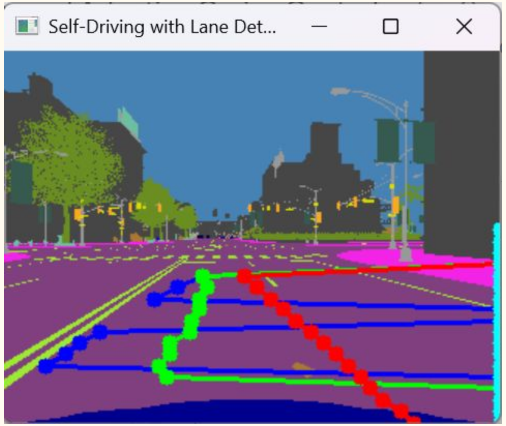
Sensor	Data	Purpose
Semantic LiDAR	Captures 3D points with object semantics (e.g., cars, pedestrians, roads)	Detects obstacles and understands the environment spatially
Camera (Semantic Segmentation)	Processed image data for the RL model (converted to NumPy array)	Captures semantic segmentation data for the environment, providing pixel-level object classification
IMU(Inertial Measurement Unit)	Measures acceleration angular velocity	Tracks vehicle dynamics
Collision Sensor	Records collision events	Detects collisions involving ego vehicle
Lane Invasion Sensor	Records lane invasion events	Detects when vehicle crosses lanes
RGB Front Camera	Captures front-facing video feed	Lane detection, object detection, and visual perception
GNSS (Global Navigation Satellite System)	Latitude, longitude, and altitude	Provides geolocation for navigation and mapping

Live Simulation Snippets

Carla Simulator Visualization

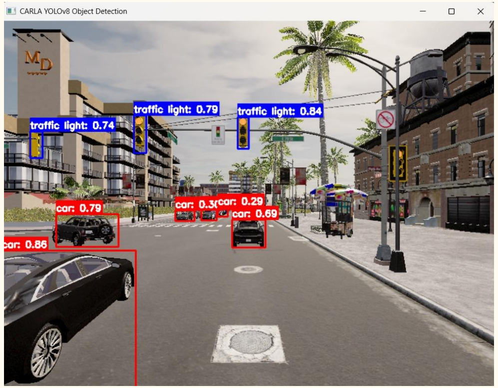


Camera -Semantic Segmentation



Camera -Lane Detection

Object Detection and Classification



Object detection helps the self-driving car recognize nearby vehicles, pedestrians, and other road elements in real-time.

- Model Used: YOLOv8
- Input: Images captured from the front-facing camera
- Output: Vehicles, people, traffic signs marked with bounding boxes.

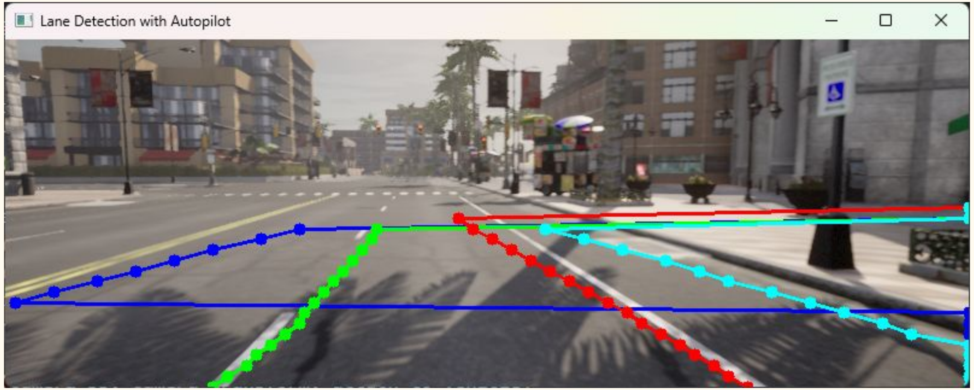
Instance Segmentation Performance Comparison (YOLOv8 vs YOLOv5)

Model Size	YOLOv5	YOLOv8	Difference
Nano	27.6	36.7	+32.97%
Small	37.6	44.6	+18.62%
Medium	45	49.9	+10.89%
Large	49	52.3	+6.73%
Xtra Large	50.7	53.4	+5.33%

*Image Size = 640

Lane Detection using Ultra-Fast-Lane-Detection

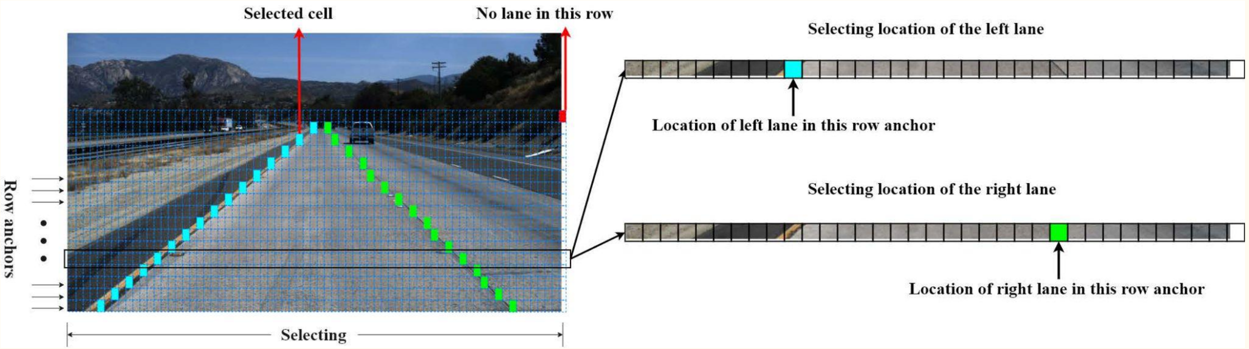
- **Model:** UFLD, a ResNet-18-based deep learning network
- **Dataset:** Trained on CULane (urban road lane markings)
- **Input:** Front-facing camera captures real-time road images
- **Output:** Detects left, center, and right lane markings per frame
- **Gridding System:** Splits image into grid cells to localize lanes
- **Post-Processing:** Softmax + custom curve-fitting algorithm
- **Visual Feedback:** Detected lanes overlaid using OpenCV
- **Performance:** Lightweight, runs at 40+ FPS, ideal for real-time use



Lane Detection using Ultra-Fast-Lane-Detection

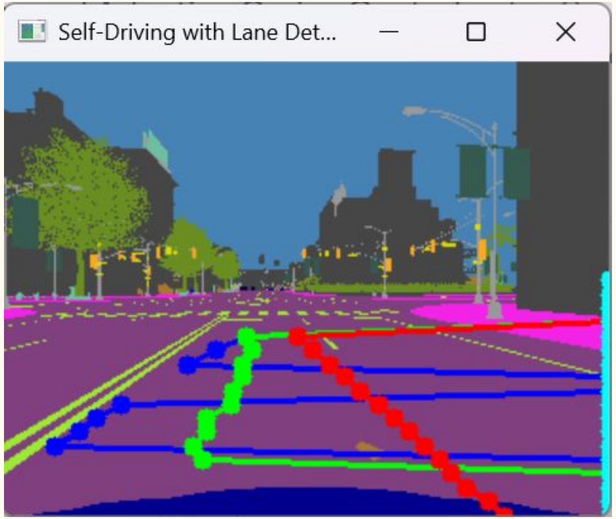
- Model: A deep learning model called Ultra-Fast-Lane-Detection (UFLD)
- Dataset: Trained on CULane – urban road lane data
- Input: A camera placed at the front of the car captures road images in real time
- Output: The model looks at each frame and detects the left, right, and center lane markings
- Performance: It's very fast ~70+ FPS and light enough to run smoothly in real-time
- Integration: Deployed in the CARLA simulator using PyTorch during both training and inference

How it Works: The model looks at each frame and detects the left, right, and center lane markings

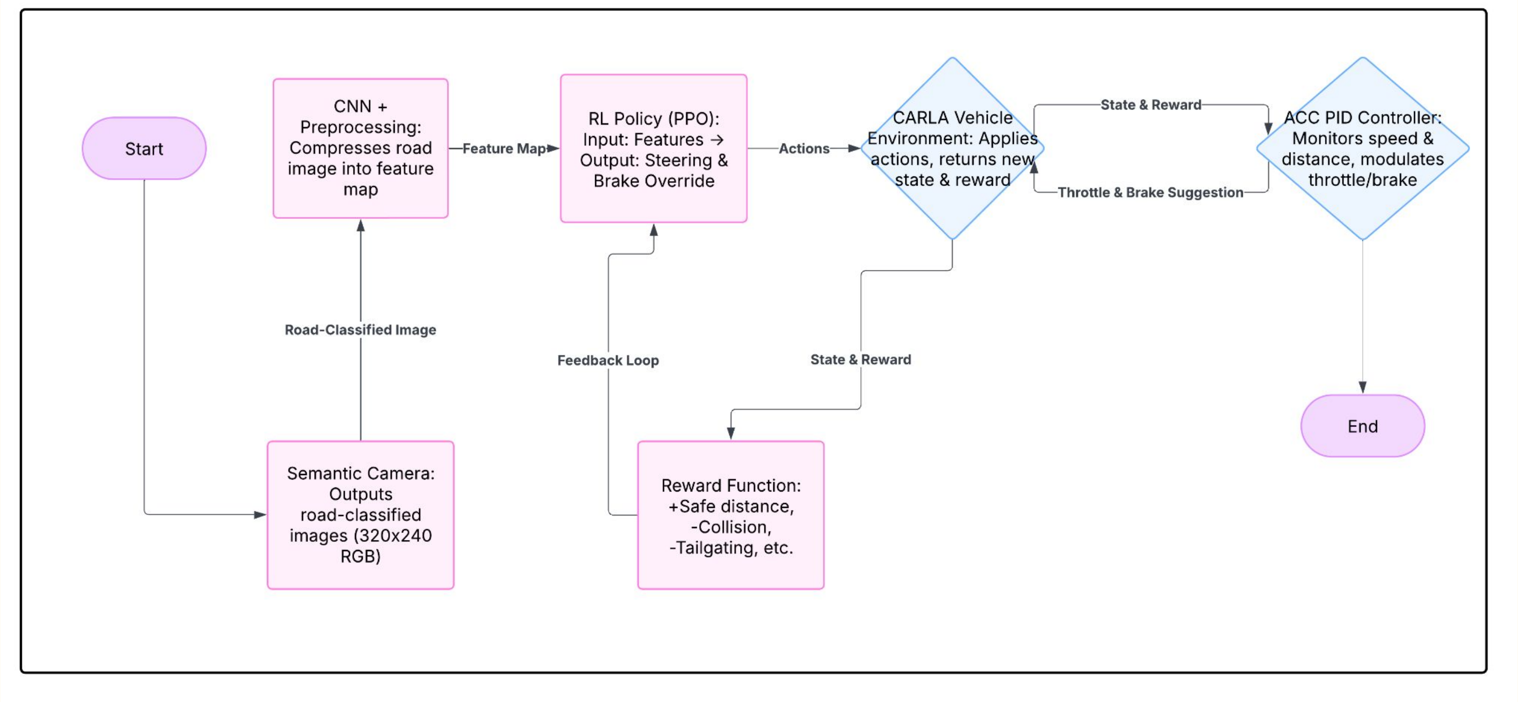


UFLD Integration

- **Model Core:** ResNet-18-based deep learning network
- **Gridding System:** Image split into grid cells; predicts lane positions per cell
- **Post-Processing:** Softmax + custom algorithm to generate smooth lane curves
- **Visual Feedback:** Lanes drawn using OpenCV for validation
- **Performance:** Fast, lightweight, and accurate — optimized for self-driving



RL Pipeline

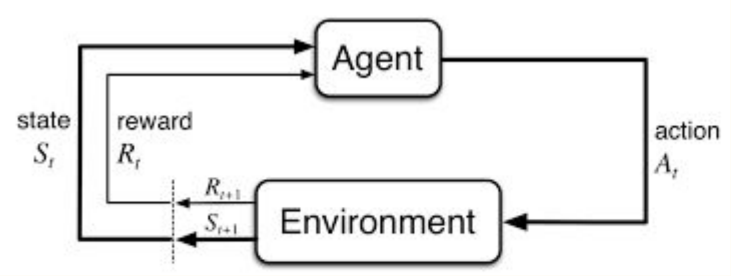

Start
CNN + Preprocessing: Compresses road image into feature map
Feature Map
RL Policy (PPO): Input: Features → Output: Steering & Brake Override
Actions
CARLA Vehicle Environment: Applies actions, returns new state & reward
State & Reward
Throttle & Brake Suggestion
ACC PID Controller: Monitors speed & distance, modulates throttle/brake
Road-Classified Image
Feedback Loop
State & Reward
End
Semantic Camera: Outputs road-classified images (320x240 RGB)
Reward Function: +Safe distance, -Collision, -Tailgating, etc.

RL Framework & Agent Design

Goal: Learn safe following behavior using Reinforcement Learning

Environment:

- Custom CarEnv(Gym-compatible)
- Inputs: CNN-processed semantic segmentation images
- Actions:
  - Steering: 9 discrete angles
  - Brake Override: [ACC, Light Brake, Full Brake]



Agent:

- PPO (Proximal Policy Optimization)
- Policy: MLP over CNN features
- Reward-driven learning without explicit rules.

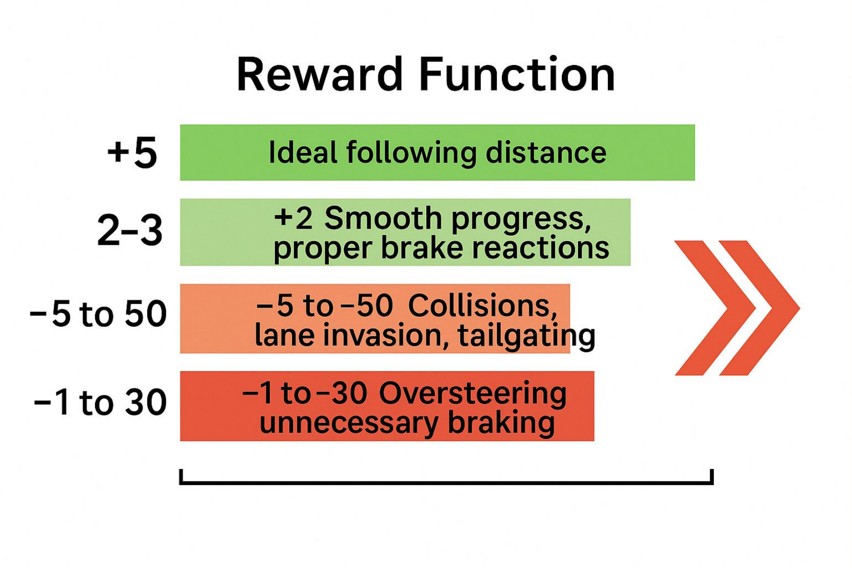
Reward Structure & ACC Logic

Reward Function:

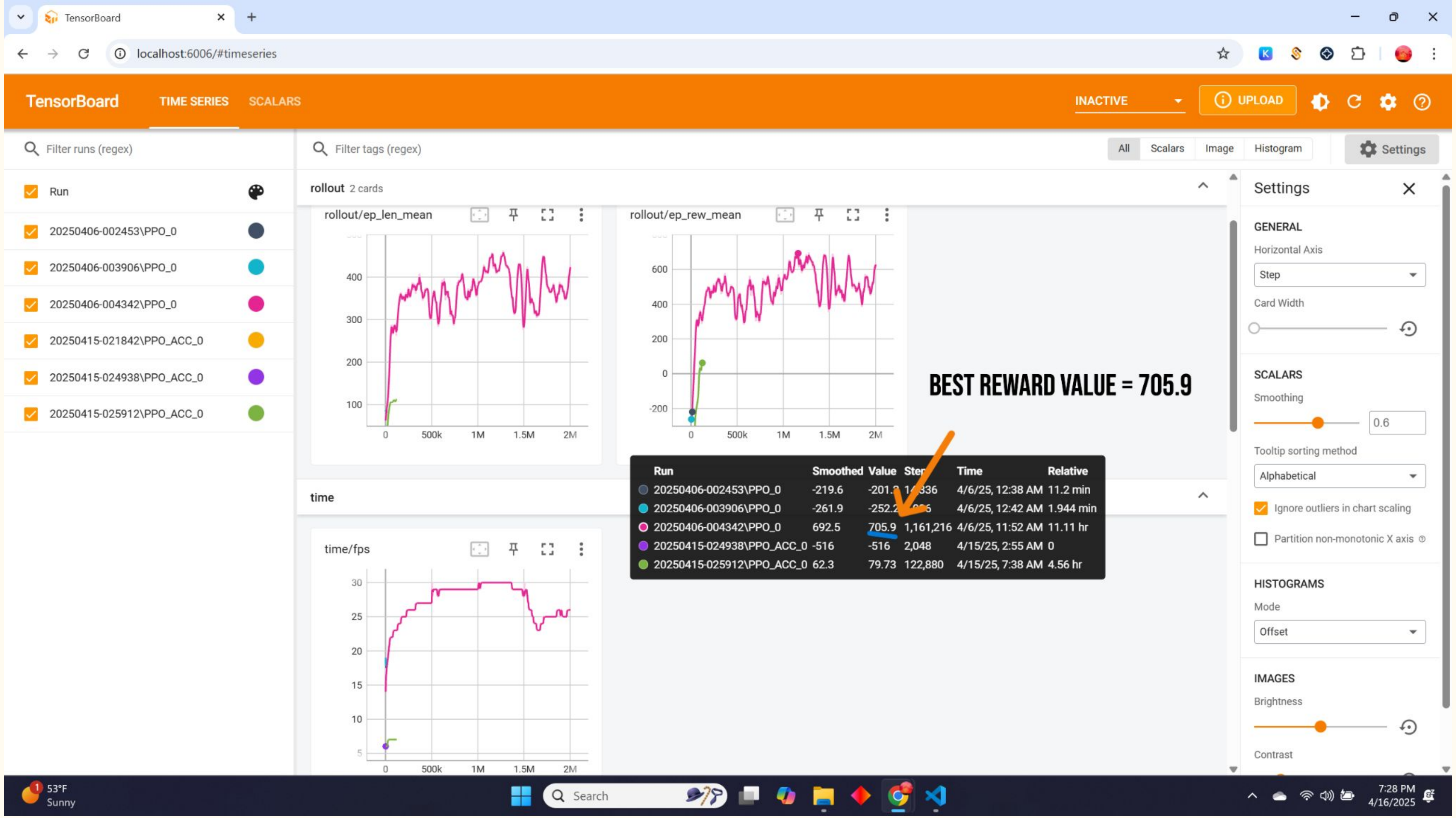
- +5 → Ideal following distance
- +2–3 → Smooth progress, proper brake reactions
- -5 to -50 → Collisions, lane invasion, tailgating
- -1 to -30 → Oversteering, unnecessary braking

ACC Integration:

- PID controllers handle throttle/brake suggestions
- RL can override with brake decisions
- ACC reward tied to real-time vehicle distance + speed



Reinforcement Learning Training Performance Analysis



Reward Curve Analysis

```
ned above are installed properly if you would like to use GPU. Follow the guide at https://www.tensorflow.org/install/gpu for how to download and setup the required
 libraries for your platform.
Skipping registering GPU devices...
Serving TensorBoard on localhost; to expose to the network, use a proxy or pass --bind_all
TensorBoard 2.10.1 at http://localhost:6006/ (Press CTRL+C to quit)
(CARLA) PS C:\Users\mynam\Downloads\Um_dearborn\Intelligent_Systems_ECE_579\Project\CARLA\Carla-Autonomous-Vehicle\carla_simulation code> python find_best_model.py
🔍 Scanning TensorBoard logs...

🏆 Best run: 20250406-004342
   ↳ Best reward: 705.9400024414062
   ↳ Step: 1161216
   ↳ Closest saved model: models/20250406-004342/model_1000000.zip
(CARLA) PS C:\Users\mynam\Downloads\Um_dearborn\Intelligent_Systems_ECE_579\Project\CARLA\Carla-Autonomous-Vehicle\carla_simulation code>
```

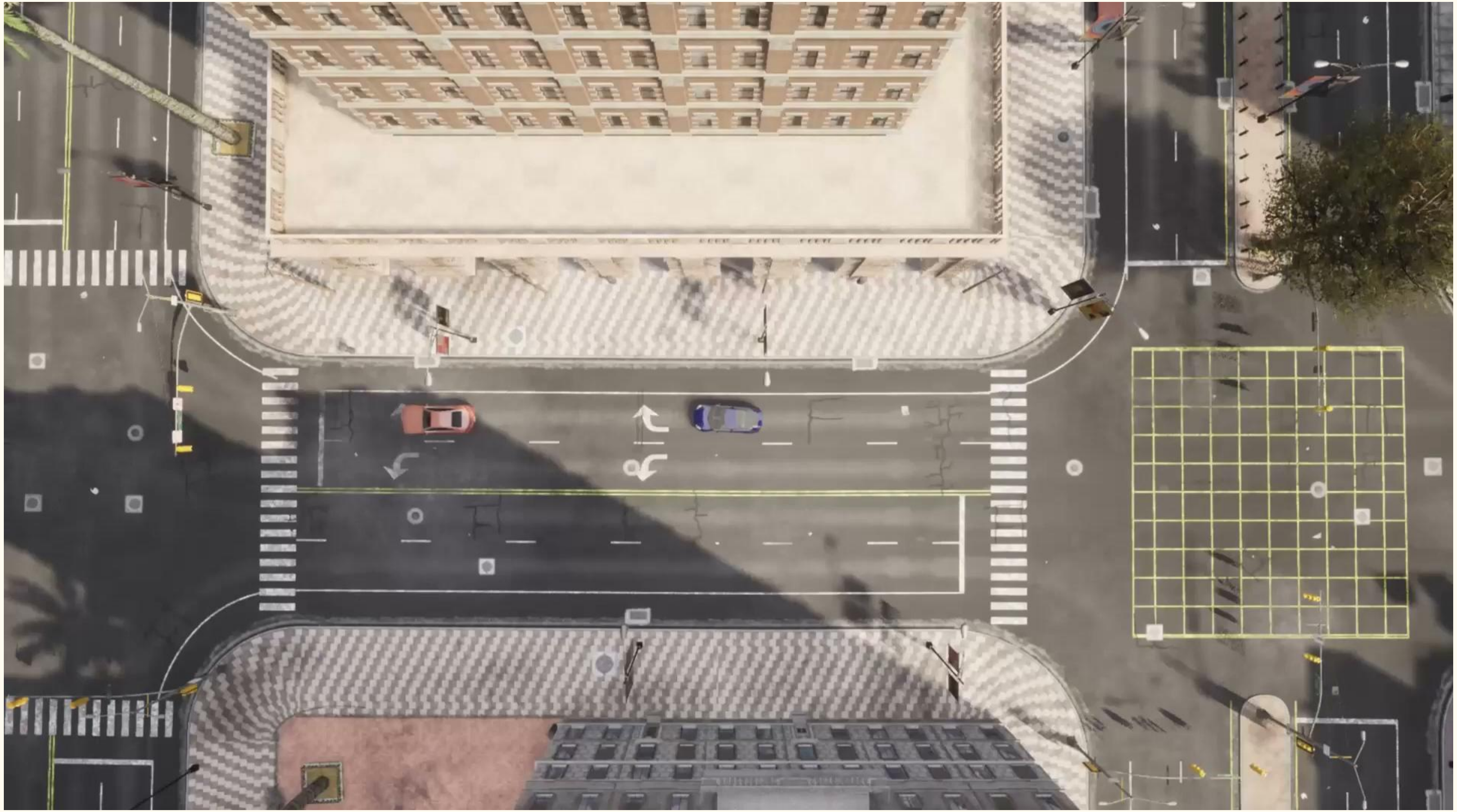
```
d.py
🔍 Scanning TensorBoard logs for reward trends...

🏆 Best run: 20250406-004342
   ↳ Peak reward: 705.94 at step 1161216

📊 Reward trend around peak:
   ↳ Step 1111216: 588.78 (Previous Checkpoint)
   ↳ Step 1161216: 705.94 (Peak)
   ↳ Step 1211216: 613.84 (Next Checkpoint)

🟢 Suggestion: Use the **model closest to the peak step** (stable or isolated peak).
(CARLA) PS C:\Users\mynam\Downloads\Um_dearborn\Intelligent_Systems_ECE_579\Project\CARLA\Carla-Autonomous-Vehicle\carla_simulation code>
```

Demo Video 1



Demo Video 2

Thank You