Interpretability in Autonomous Driving: Visual Attribution Analysis of RL Agents

- Anubhav Paras

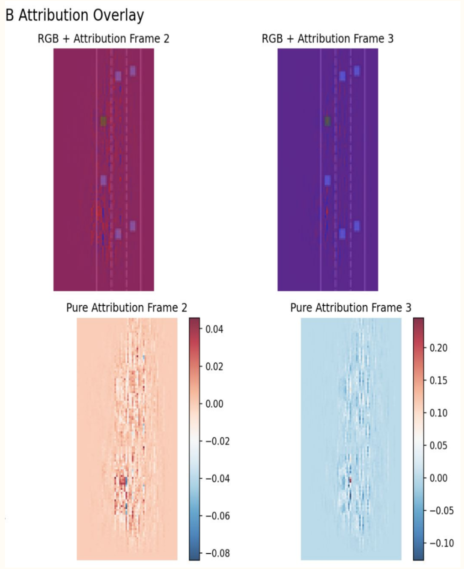
Interpretability in Autonomous Driving: Visual Attribution Analysis of RL Agents

Understanding What Deep RL Agents See When Making Driving Decisions

Motivation: Autonomous driving systems must be interpretable to ensure safety and trust. Understanding what visual features influence RL agent decisions is crucial for deployment.

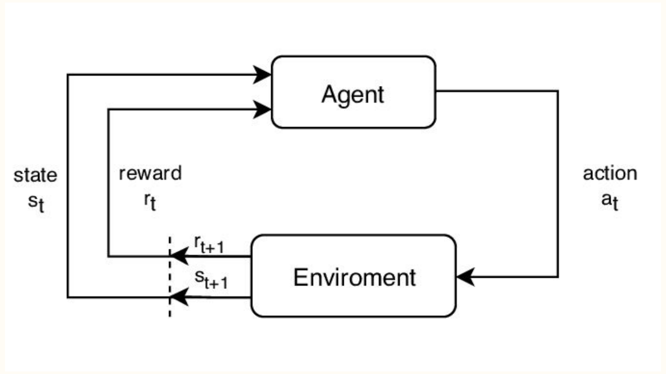
Main Idea: Train a Deep RL agent on highway driving tasks, then use gradient-based attribution methods to visualize which pixels influence each driving decision.

Results: Successfully identified key visual patterns the agent focuses on (lane markings, nearby vehicles, road boundaries) and revealed decision-making transparency through pixel-level attribution maps.

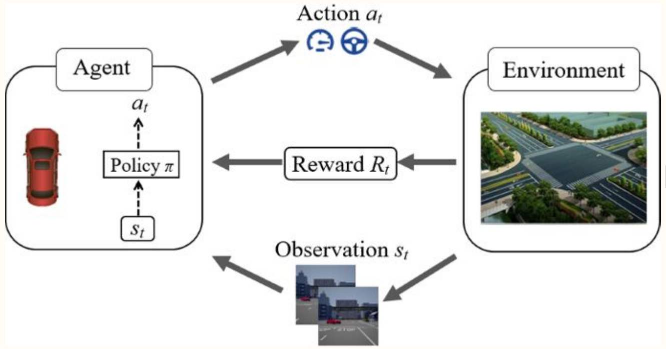


Literature Review: Reinforcement Learning (RL)

Reinforcement Learning



Reinforcement
In Autonomous Driving



ng

Literature Review: Interpretability in Deep RL

- Visual Attribution Methods
  - Gradient-based methods: Saliency maps, Integrated Gradients (Sundararajan et al., 2017).
  - Perturbation-based: LIME, SHAP for explaining individual predictions.
  - Attention mechanisms: Built-in interpretability through attention weights.

- RL Interpretability Challenges
  - Sequential decisions: Actions affect future states and rewards.
  - Temporal dependencies: Understanding long-term strategy vs. immediate reactions.
  - High-dimensional observations: Raw pixel inputs make interpretation complex.

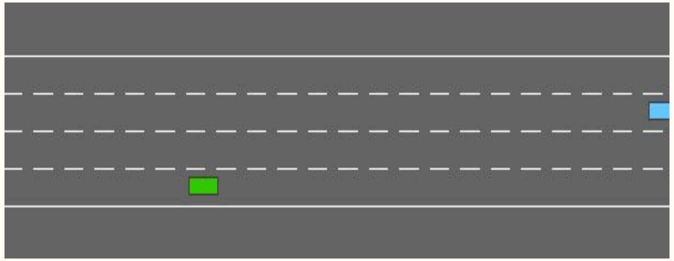
Literature Review: Interpretability in Deep RL

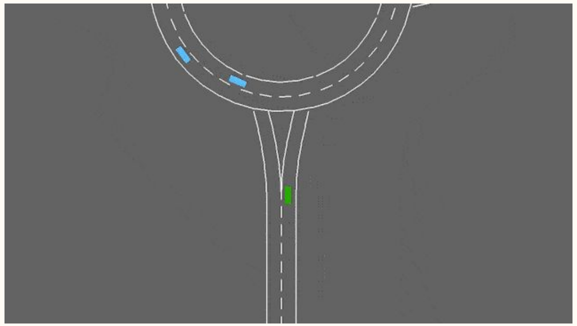
- Automotive Applications
  - Safety-critical systems: Requirement for explainable AI in autonomous vehicles.
  - Regulatory compliance: EU AI Act and similar regulations demanding transparency.
  - Trust and adoption: User acceptance depends on understanding system behavior.

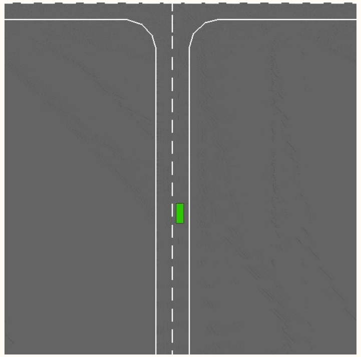
Approach: Environment & Training

Highway-Env Simulation

A collection of environments for autonomous driving and tactical decision-making tasks.


highway-env


roundabout-env


intersection-env

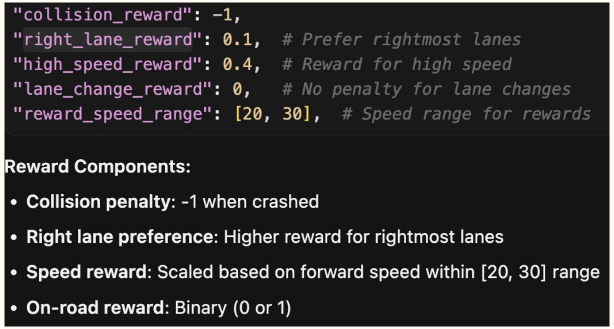
Approach: Environment & Training

Highway-Env Simulation

State space: RGB image observations (4-stacked frames).

Action space: Discrete actions (accelerate, brake, turn left/right, idle)

Reward structure: highway-env

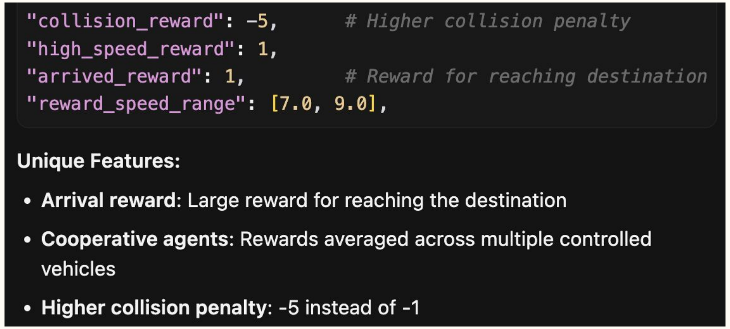


```
"collision_reward": -1,
"right_lane_reward": 0.1,  # Prefer rightmost lanes
"high_speed_reward": 0.4,  # Reward for high speed
"lane_change_reward": 0,   # No penalty for lane changes
"reward_speed_range": [20, 30],  # Speed range for rewards
```

Reward Components:

- **Collision penalty**: -1 when crashed
- **Right lane preference**: Higher reward for rightmost lanes
- **Speed reward:** Scaled based on forward speed within [20, 30] range
- **On-road reward**: Binary (0 or 1)

intersection-env



```
"collision_reward": -5,      # Higher collision penalty
"high_speed_reward": 1,
"arrived_reward": 1,         # Reward for reaching destination
"reward_speed_range": [7.0, 9.0],
```

Unique Features:

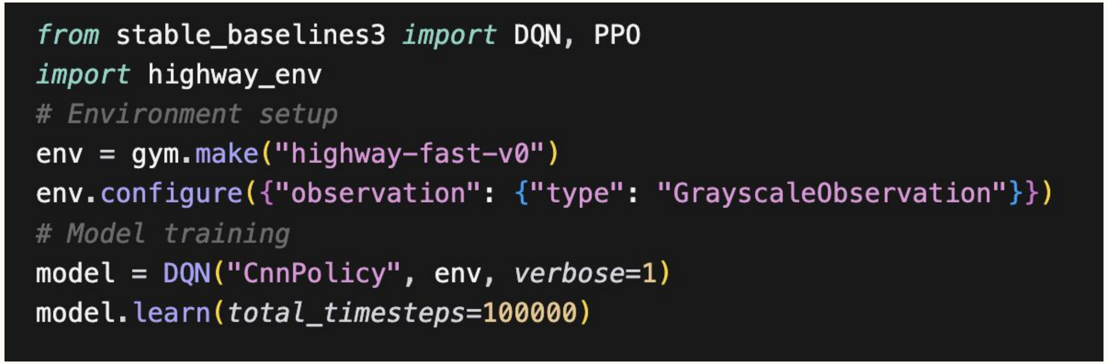
- **Arrival reward**: Large reward for reaching the destination
- **Cooperative agents**: Rewards averaged across multiple controlled vehicles
- **Higher collision penalty**: -5 instead of -1

Traffic scenarios: Multi-lane highway with dynamic traffic.

Approach: Environment & Training

RL Training

- DQN Algorithm: Deep Q-Network with experience replay and target networks.
- PPO Alternative (for continuous action space): Proximal Policy Optimization for policy gradient approach.
- stable-baselines3: Implementation framework with optimized hyperparameters.
- CNN Architecture: Convolutional layers for visual feature extraction.

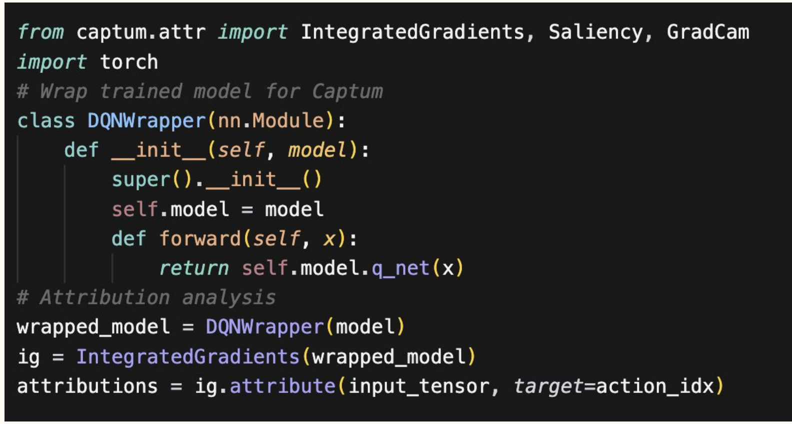


```python
from stable_baselines3 import DQN, PPO
import highway_env
# Environment setup
env = gym.make("highway-fast-v0")
env.configure({"observation": {"type": "GrayscaleObservation"}})
# Model training
model = DQN("CnnPolicy", env, verbose=1)
model.learn(total_timesteps=100000)
```

Approach: Interpretability Analysis

Captum Integration

- Attribution Methods Applied
  - Integrated Gradients: Path-based attribution with baseline comparison.
  - Integrated gradients is a simple, yet powerful axiomatic attribution method that requires almost no modification of the original network.
  - Captum provides a generic implementation of integrated gradients that can be used with any PyTorch model.



```
from captum.attr import IntegratedGradients, Saliency, GradCam
import torch
# Wrap trained model for Captum
class DQNWrapper(nn.Module):
    def __init__(self, model):
        super().__init__()
        self.model = model
        def forward(self, x):
            return self.model.q_net(x)
# Attribution analysis
wrapped_model = DQNWrapper(model)
ig = IntegratedGradients(wrapped_model)
attributions = ig.attribute(input_tensor, target=action_idx)
```

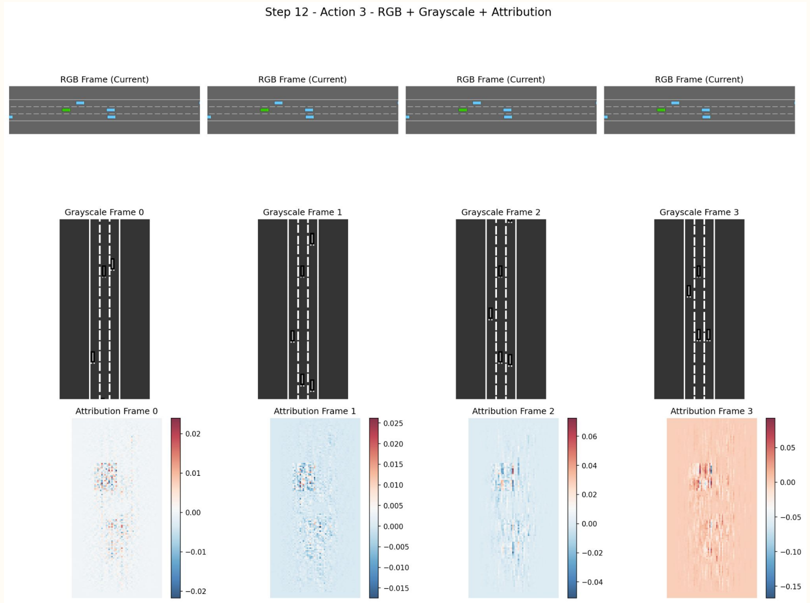
Approach: Interpretability Analysis

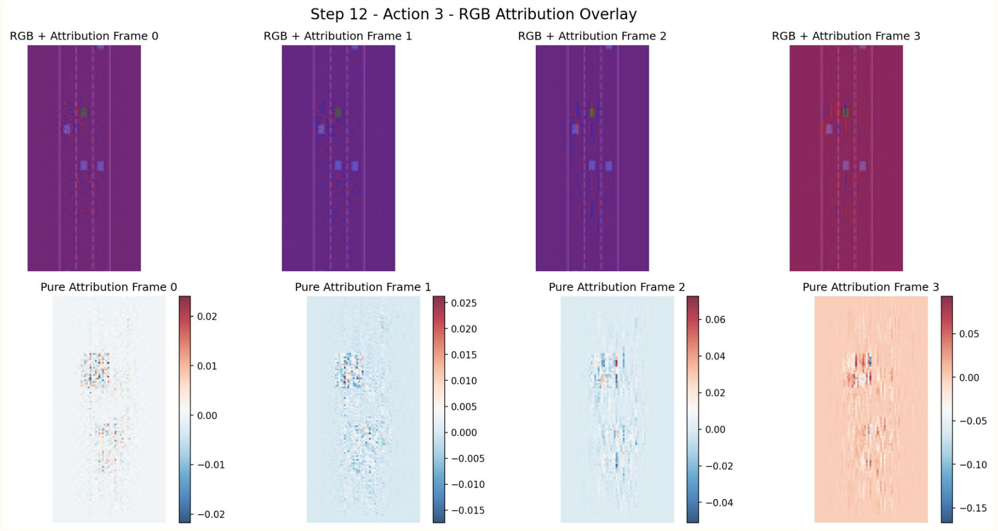
Captum Integration

- Analysis Framework
  - Frame-by-frame analysis: Attribution maps for each decision point
  - Action-specific attribution: Different visualizations per action type

Results

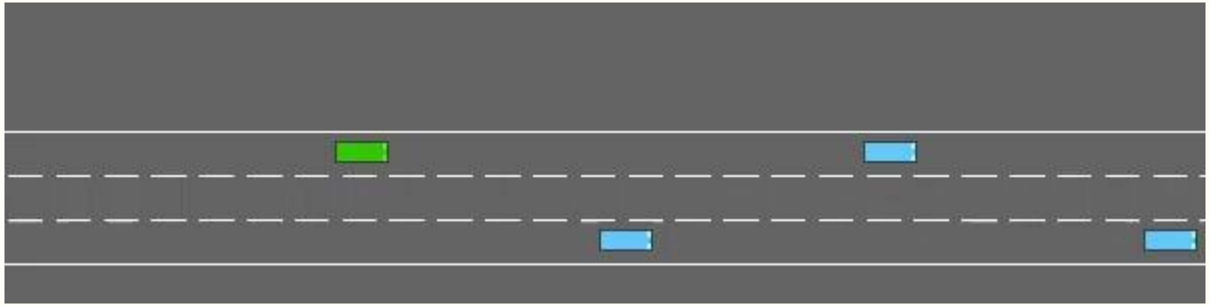
The trained agent demonstrates focusing on relevant driving cues and showing context-appropriate visual prioritization across different driving scenarios.





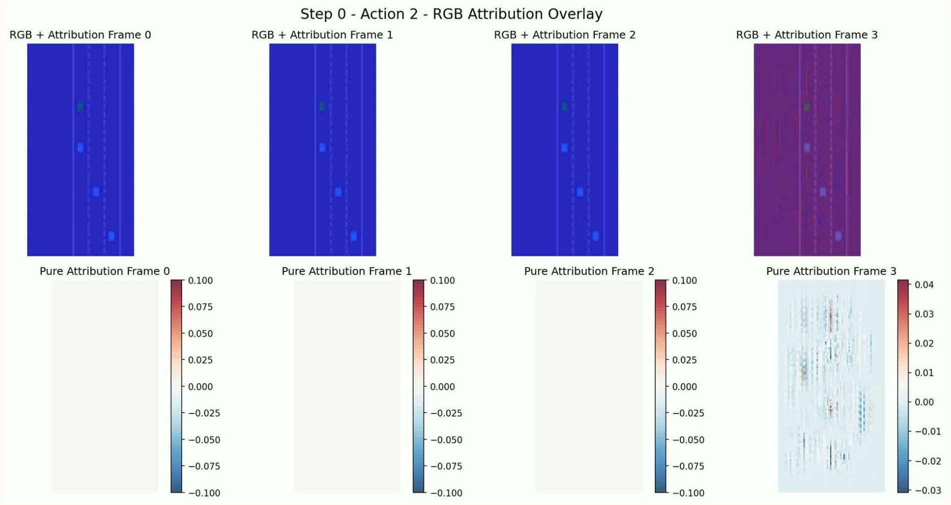
Results

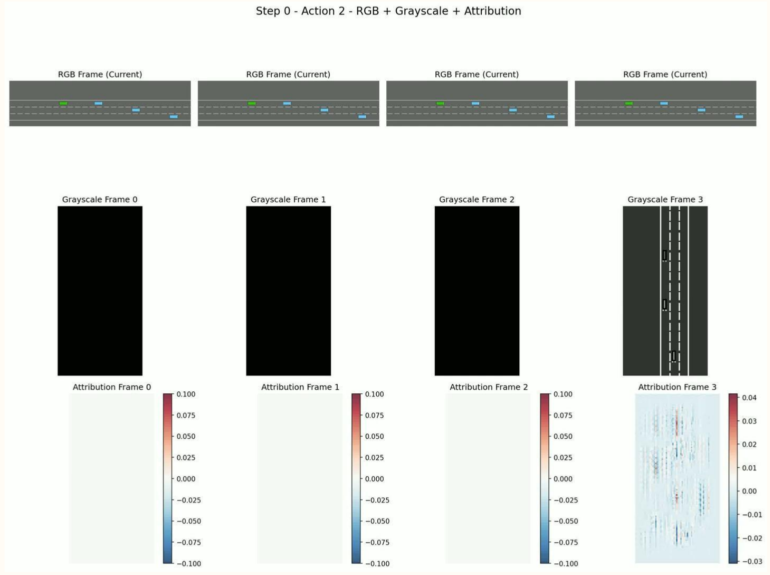
Trained DQN agent on highway-env.



Results

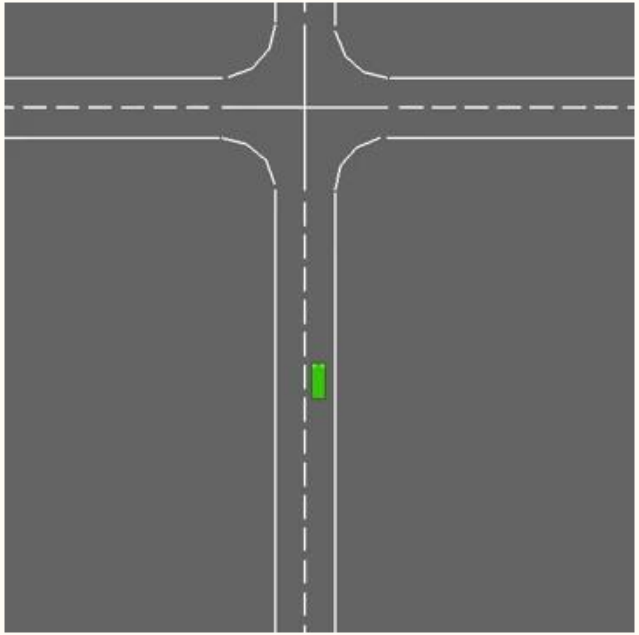
Trained DQN agent on highway-env: Attribution analysis





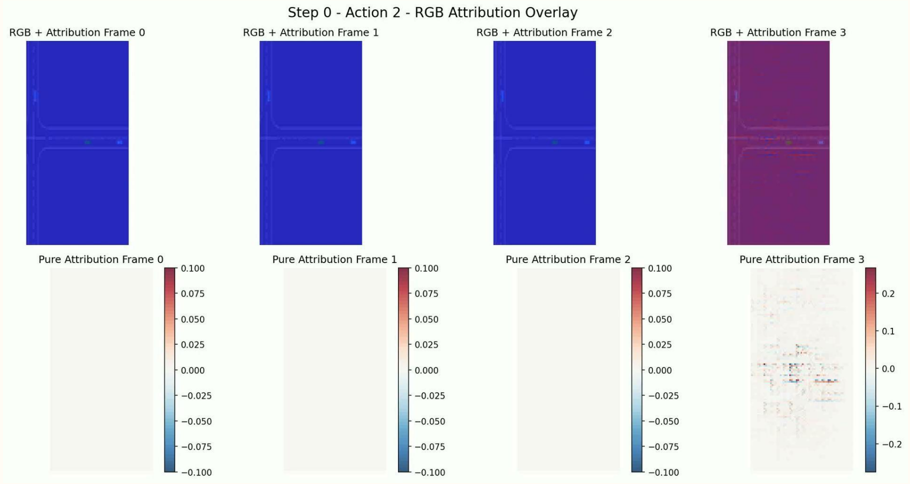
Results

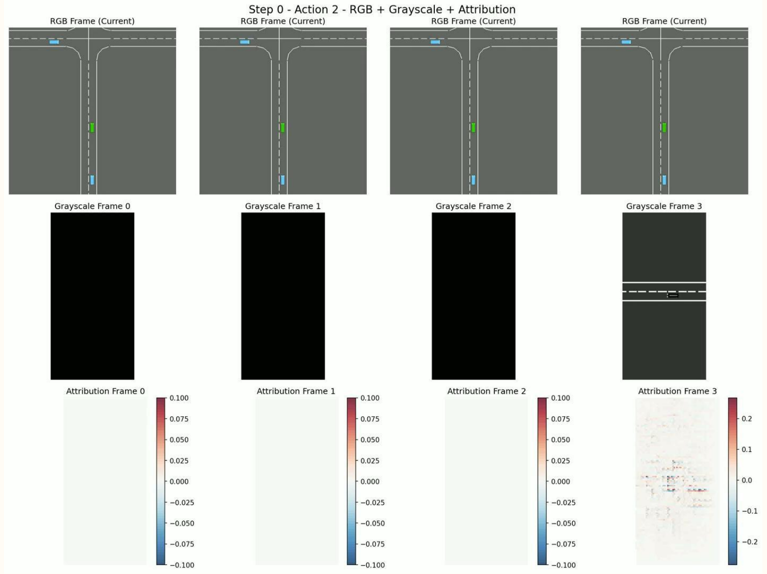
Trained DQN agent on intersection-env.



Results

Trained DQN agent on intersection-env: Attribution analysis





Technical Implementation Details

- Frameworks & Libraries
  - Highway-Env: https://github.com/eleurent/highway-env
  - Stable-Baselines3: https://stable-baselines3.readthedocs.io/
  - Captum: https://captum.ai/
  - PyTorch: https://pytorch.org/
- Training Configuration
  - Episodes: 100,000 training steps
  - Network: stable-baselines3 CNNPolicy
  - Hyperparameters: Learning rate 5e-4, batch size 32
  - Hardware: NVIDIA GPU 3060 - 12GB VRAM
  - Training time: ~1hr - 1.5hrs
- Github
- Demo

Future Research Directions

- Multi-Modal Analysis:
  - This was a simple experiment with images as modality.
  - Incorporate radar, lidar data interpretability.
- Temporal Attribution:
  - Long-term decision influence analysis.
- Human Studies:
  - Compare agent vs. human attention patterns.
- Real Vehicle Testing:
  - Validation on actual autonomous systems.
- Integration with VLMs:
  - Textual /natural language explanations of the attributions.

References

- Sundararajan, M., Taly, A., & Yan, Q. (2017). Axiomatic attribution for deep networks. International conference on machine learning (pp. 3319-3328).
- Mnih, V., Kavukcuoglu, K., Silver, D., et al. (2015). Human-level control through deep reinforcement learning. Nature, 518(7540), 529-533.
- Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347.
- Kokhlikyan, N., Miglani, V., Martin, M., et al. (2020). Captum: A unified and generic model interpretability library for PyTorch. arXiv preprint arXiv:2009.07896.
- Leurent, E. (2018). An environment for autonomous driving decision-making. GitHub repository: highway-env.
- Raffin, A., Hill, A., Gleave, A., et al. (2021). Stable-Baselines3: Reliable reinforcement learning implementations. Journal of Machine Learning Research, 22(268), 1-8.
- Selvaraju, R. R., Cogswell, M., Das, A., et al. (2017). Grad-cam: Visual explanations from deep networks via gradient-based localization. Proceedings of the IEEE international conference on computer vision (pp. 618-626).
- Shrikumar, A., Greenside, P., & Kundaje, A. (2017). Learning important features through propagating activation differences. International conference on machine learning (pp. 3145-3153).